UNIVERSIDAD CARLOS III DE MADRID

TRABAJO FIN DE GRADO

DETECTOR DE SONIDOS FAMILIARES MEDIANTE TELÉFONOS MÓVILES EN ENTORNOS ADVERSOS

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Autor: Jorge Homobono Rubio

Tutor: Raúl Sánchez Reíllo

Leganés, Septiembre de 2014

Agradecimientos

A mis padres, por su apoyo tanto moral como económico a lo largo de estos años, sin ellos la realización de este trabajo hubiera sido imposible. A mi hermana Laura, por su infinita paciencia. A Mónica por ser capaz de ayudarme y motivarme en todo momento. Y especialmente a mis abuelos que tanto se preocupan por mi futuro.

Asimismo quería agradecer a Raúl y a los miembros del GUTI la posibilidad de dar a los estudiantes de introducirse en el mundo de las tecnologías identificación, en mi caso con la plataforma Android. Quería destacar también, a mis compañeros de trabajo en CAF que tanto me han enseñado este último año, todo lo aprendido de ellos ha dado una nueva dimensión a este TFG, haciéndolo más ordenado y completo.

En definitiva, a todos aquellos familiares, amigos y compañeros que durante estos años han sido de ayuda para la consecución de mis metas.

Resumen

Los *smartphones* o "teléfonos inteligentes" han irrumpido en los últimos años, de manera que han cambiado completamente el uso que se les había dado hasta ahora a los teléfonos móviles. Estos nuevos terminales tienen potentes hardware gestionados por novedosos Sistemas Operativos, como *Android,* que han permitido la inclusión de aplicaciones capaces de realizar multitud de tareas.

Por otra parte, las ayudas técnicas a la sordera han evolucionado mucho desde la llegada de la era digital. Aunque estas ayudas técnicas se han centrado en el uso y mejora de los audífonos, esta nueva era digital también ha propiciado la aparición de otro tipo de ayudas como detectores de sonidos que son complementarios al uso de los audífonos.

En este Trabajo Fin de Grado, se ha desarrollado una aplicación móvil para *Android* que es capaz de detectar sonidos en el entorno del hogar con la finalidad de ayudar a personas con problemas de audición. La aplicación se encarga de mandar una vibración y un mensaje iluminado cuando detecte un sonido, de esta manera el usuario con problemas de audición podrá identificar el sonido.

Abstract

The use of Smartphones has become very popular in the last few years. The new designs and functionalities added, as well as the incorporation of powerful hardware and complex and specialized Operating Systems (OS) in the devices have completely changed the way of using them. They contain multiple kinds of applications with very different uses and tasks.

Moreover, the deafness assistive technologies have improved notably since the arrival of the digital era. Though deafness assistive systems have focused in headset development, the digital technologies have allowed other assistive technologies to come out. These new technologies are complementary with the traditional headsets.

This document explains the development of an Android application which can detect different house sounds including the ones made by a microwave, a door bell, an intercom, a fridge sound or a telephone. The purpose of this project is to help deaf people in their homes sending vibration alerts and screen notifications whenever one of these sounds is detected.

Índice

Índice de Figuras

Índice de Tablas

Listado de Acrónimos

ADT	Android Development Tools
API	Application Programming Interface
CD	Compact Disc
DSP	Digital Signal Processing
DVD	Digital Versatile Disc
EDGE	Enhaced Data rates for GSM Evolution
FFT	Fast Fourier Transform
GPRS	General Packet Radio Service
GSM	Global System for Mobile
GUI	Graphical User Interface
IDE	Integrated Development Enviroment
MMS	Multimedia Message Service
NDK	Native Development Kit
NFC	Near Field Communication
NMT	Nordic Mobile Telephony
PCM	Pulse Code Modulation
SDK	Software Development Kit
SMS	Short Message Service
SO	Sistema Operativo

TFG	Trabajo Fin de Grado
UMTS	Universal Mobile Telecommunications System
VHS	Video Home System
XML	eXtensible Markup Language

1 Introducción

En la presente memoria se va a exponer un TFG cuya finalidad es crear una aplicación móvil que sea capaz de reconocer sonidos en el entorno del hogar para personas con problemas de audición.

1.1 Motivación

Desde la salida al mercado de los primeros teléfonos móviles en los años 80 [1], estos han ido incorporando nuevas tecnologías, que han ido proveyendo poco a poco a los terminales de novedosas funciones, dejando en la actualidad a un lado su función inicial: realizar llamadas.

Los teléfonos móviles han ido mejorando su capacidad de procesamiento poco a poco hasta lograr realizar operaciones que antes solo podían realizar los ordenadores, con una gran ventaja respecto a estos últimos: su tamaño es mucho menor. Aunque la potencia de los ordenadores siga siendo superior, el tamaño de un móvil da la posibilidad al usuario de portar el dispositivo en cualquier momento.

En los últimos años, han irrumpido en el mercado los denominados *smartphones* [2] (teléfonos inteligentes), cuya principal novedad es la inclusión de una gran cantidad de programas o aplicaciones que explotan toda la capacidad de cálculo de la que dispone el *hardware* de estos dispositivos.

Todas estas mejoras de *hardware* y *software* han provocado que, gracias a su tamaño y su potencia, estos dispositivos ofrezcan multitud de posibilidades que han revolucionado nuestras vidas: las redes sociales, la posibilidad de ver en *streaming* su película o serie favorita, comprobar el tiempo que va a hacer el día siguiente o simplemente mantener una conversación con un familiar al que le separan miles de kilómetros. Por todos estos motivos los *smartphones* se han hecho muy populares en la actualidad, logrando en España una tasa de penetración que supera el 60% [3].

Claro está que la inclusión de estas nuevas aplicaciones móviles ha sido determinante, pero la aparición de nuevos SO pensados para estos pequeños dispositivos también lo ha sido. Los nuevos SO móviles son mucho más ligeros que los que usan los ordenadores y su diseño está centrado en la conectividad y optimización de recursos.

Dentro de los diferentes SO que existen para estos *smartphones*, este trabajo se realiza en *Android*, que a día de hoy domina claramente el mercado con una cuota de mercado mundial del 79% en el año 2013.

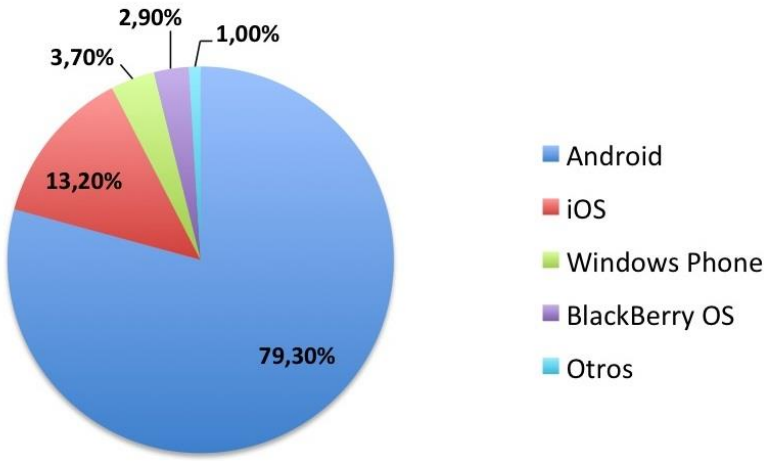


Figura 1. Cuota de Mercado Mundial de Teléfonos Móviles por SO en el año 2013 [4].

Esta plataforma ha ganado mucho peso en los últimos años, extendiéndose a nuevos dispositivos como las *tablets* y con intención de llegar a otros como televisores, relojes o automóviles, por lo que posee un enorme un potencial con vistas al futuro.

Dada la gran popularidad de *Android*, el potencial que posee y el extenso soporte que goza el desarrollador, resulta realmente interesante adentrarse en esta novedosa plataforma que hará al nuevo programador de *Android* ampliar sus horizontes como desarrollador además de mejorar sus perspectivas profesionales.

Por otro lado, en el campo de las ayudas a la audición, como ya es sabido, se han logrado multitud de avances en los últimos años en los audífonos como se detallará en el apartado 2.4 del Estado del Arte. También existen otra corriente de dispositivos de ayuda a la audición como el *PocketVib Cordless Vibrator* [5] que ayudado de otro dispositivo vibra y se ilumina al detectar sonidos en el hogar como el timbre, la alarma de incendios, el sonido del teléfono, etc. Dispositivos como este han servido de inspiración para el desarrollo de la aplicación del presente TFG.

Figura 2. PocketVib cordless vibrator [5].

Actualmente la existencia de aplicaciones móviles que puedan ser de utilidad para personas con problemas de audición es escasa. Por eso resulta interesante la posibilidad de desarrollar aplicaciones en campos a los que, hasta ahora no se ha profundizado.

1.2 Objetivo del Proyecto

El objetivo principal del proyecto es conseguir desarrollar una aplicación *Android* que sea capaz de detectar sonidos en el entorno del hogar para personas con problemas de sordera. La aplicación será capaz de detectar sonidos como el timbre de una casa, el timbre de un portal, el microondas, el teléfono o el sonido de una nevera abierta. Dicho objetivo principal se puede dividir en los siguientes sub-objetivos:

- Familiarización con una nueva plataforma, *Android* y su entorno de desarrollo *Eclipse*, completamente novedosos respecto a lo aprendido durante el Grado.
- Grabar audio con un dispositivo *Android* en formato sin comprimir.
- Aprendizaje en técnicas de procesado digital de audio. En particular utilizando la plataforma *Android*.
- Iniciación en el uso de patrones para identificar sonidos.
- Construir una aplicación *Android* sencilla de utilizar para el usuario. Dado que los problemas de sordera suelen aparecer en personas en edades avanzadas, se perseguirá la sencillez tanto en el interfaz gráfico como en la usabilidad de la aplicación.
- Lograr una aplicación final que sea eficiente en el uso de recursos del dispositivo, teniendo en cuenta que el procesado de audio requiere una gran capacidad de procesamiento.

1.3 Entorno Socio-Económico

En el plano Socio-Económico la aplicación puede ser de gran utilidad para todas aquellas personas que tengan problemas de audición y por algún motivo, no pueden usar los clásicos audífonos o simplemente requieren esta aplicación como una alternativa complementaria a sus audífonos.

Desde ese punto se podría explotar el alcance mundial que tiene el SO *Android* para llegar a un gran número de potenciales usuarios que pudieran requerir de la presente aplicación.

1.4 Estructura del Documento

La presente memoria ha sido estructurada en los siguientes capítulos:

- **Introducción**: En este capítulo se comentan los aspectos generales del proyecto y se presenta el trabajo a realizar.
- **Estado del Arte**: Se realiza una introducción a la telefonía móvil y al estado actual de los *smartphones*. Posteriormente se muestra la evolución en el registro y procesado de audio, para finalizar con la situación actual de las ayudas a la sordera.
- **Entorno de Desarrollo**: Expone las diferentes tecnologías que han sido necesarias para desarrollar la aplicación.
- **Diseño de la Aplicación**: En el diseño se describen los requisitos que cumple la aplicación y cómo se ha llegado al diseño de la solución final de la misma después de manejar distintos planteamientos.
- **Desarrollo**: Retrata en qué ha consistido el proceso de desarrollo: tanto los problemas surgidos a la hora de implementar lo propuesto en el diseño, como las soluciones encontradas.
- **Pruebas**: Se listan todas las pruebas a las que ha sido sometida la aplicación al finalizar su desarrollo.
- **Conclusiones y Líneas Futuras**: En este capítulo se recogen las conclusiones después de finalizar la aplicación, resumiendo qué conceptos han sido los más determinantes. Se hace también una exposición de las posibles mejoras de la aplicación y líneas futuras.
- **Bibliografía**: Finalmente, se lista toda la bibliografía utilizada a lo largo de la redacción de la memoria.

Además se han incorporado dos anexos: El *Anexo A* con la planificación y presupuesto, y el *Anexo B* con un breve manual de usuario de la aplicación desarrollada para este TFG.

2 Estado del Arte

En este capítulo se resumirá la historia de la telefonía móvil y su evolución. Se hablará de los diversos SO existentes para dispositivos móviles y el alcance de sus aplicaciones hoy en día. A continuación, se realizará una breve exposición del registro y procesado del sonido, y finalmente, se mostrarán brevemente las ayudas técnicas a la sordera que existen actualmente.

2.1 Historia de la Telefonía Móvil

Los primeros precedentes de la telefonía móvil se remontan a la Segunda Guerra Mundial cuando el ejército americano veía la necesidad de comunicarse a distancia en el campo de batalla.

Figura 3. *Motorola Handie Talkie H12-16* [6].

Para satisfacer estas necesidades Motorola lanza el *Handie Talkie H12-16* que permite el contacto con las tropas vía ondas de radio cuya banda de comunicación no supera los 600 kHz. [7]

Poco después, en la década de los años 1940, se incorporan estos dispositivos a los automóviles gracias a ingenieros de los Laboratorios Bell.

Estos primeros dispositivos eran voluminosos y consumían mucha potencia, solo admitían un escaso número de conversaciones simultáneas y ofrecían un área de cobertura muy limitada.

En los años 60, Leonid Kupriyanovich, un ingeniero de Moscú, diseñó un terminal pequeño que cabía en la palma de la mano. El dispositivo lograba realizar llamadas con tenía un alcance de más de 30 Km.

Poco a poco fueron apareciendo más prototipos hasta que en marzo del año 1984 se lanzó el primer teléfono móvil al mercado, en concreto fue el modelo *Motorola DynaTAC 8000x* [1].

Figura 4. *Motorola DynaTAC 8000x. El primer teléfono móvil de la historia* [1].

El *Motorola DynaTAC 8000x* se lanzó con un precio de salida de 3.995 dólares, pesaba cerca de 1kg, tenía un tamaño de 33,02 x 4,45 x 8,89 centímetros y su batería duraba una hora en llamada u ocho horas en espera. Además poseía una pequeña pantalla de LED monocolor.

Aunque Motorola había desarrollado el *DynaTAC 8000x* para uso interno de la propia compañía, logró tener un número de ventas notable, a pesar del elevado precio del dispositivo: un año más tarde de su lanzamiento se habían vendido 300.000 unidades.

El éxito cosechado por el *DynaTAC 8000x* propició que aparecieran nuevos terminales, que aunque parecieran aparatosos para los estándares actuales, fueron una revolución para la época ya que podían ser utilizados por una única persona.

Para la utilización de estos terminales era necesario el uso de una red, por eso el fabricante Ericsson había lanzado en el año 1981 el sistema NMT 450, el primer sistema del mundo de telefonía móvil. Este sistema seguía utilizando canales de radio analógicos como sus predecesores pero llegaba a frecuencias de 450 Mhz con modulación en frecuencia (FM). En 1986, Ericsson modernizaría su sistema llevándolo hasta el nivel NMT 900, lo cual permitió dar servicio a un mayor número de usuarios.

En los años 90 nace lo que se conoce como Segunda Generación o 2G, en esta nueva generación mejora la calidad de la voz y aparece la transmisión de datos a baja velocidad. Aunque la novedad más significativa, es la digitalización de las comunicaciones.

Gracias a la digitalización se permite, además de la mejora de la voz, aumentar significativamente el nivel de seguridad y especialmente simplificar la fabricación del terminal, disminuyendo el tamaño de los mismos y provocando que los costes de fabricación disminuyeran notablemente, aumentando así la venta de terminales.

Con esta segunda generación aparece el estándar GSM (*Global System for Mobile*) en Europa con frecuencias de 900 y 1.800 Mhz. Posee las siguientes características:

- Buena calidad de voz, debido al procesado digital.
- Aparición del SMS (Short Message Service)
- Itinerancia. Posibilidad de utilizar el dispositivo fuera de la red de origen.
- Deseo de implantación internacional.
- Terminales realmente portátiles (de reducido peso y tamaño) a un precio asequible.
- Instauración de un mercado competitivo con multitud de operadores y fabricantes.

Dada la baja velocidad de transferencia que existía en la 2G (tan sólo 9.6 Kbps), entre la segunda y la tercera generación hay una generación de transición conocida como 2.5G en la que se mejora el servicio de mensajería. Aparece la mensajería multimedia (MMS), apareciendo con ella la posibilidad de enviar videos, imágenes o textos gracias a la tecnología GPRS, que permite velocidades de datos desde 56 kbit/s hasta 114 kbit/s. Aparece también la tecnología EDGE que permite aumentar la velocidad de transferencia llegando a 384 kbit/s.

La tercera generación (3G) surge a en los años 2000 y es en la cual la telefonía móvil evoluciona más significativamente. La velocidad de transmisión vuelve a aumentar gracias al nuevo sistema UMTS permitiendo velocidades de 7.2 Mbit/s. Estas elevadas velocidades permiten la aparición de nuevos servicios como navegar por internet, la descarga de contenidos, servicios de videollamada, mensajería instantánea, la utilización del correo electrónico, etc.

Con la tercera generación surge la necesidad de manejar un alto volumen de información debido a los nuevos servicios provistos gracias al sistema UMTS. Para ello, aparecen los llamados *Smartphones* (teléfonos inteligentes) que pueden llegar a cumplir las mismas funciones que un ordenador personal. De esta generación hay que destacar los sistemas operativos de estos smartphones siendo los más importantes *Android*, *Symbian* y *Windows Phone*. Muy en relación con los smartphones están las redes sociales tales como *Facebook*, *Twitter* o *Tuenti* entre otras, o aplicaciones como *Whatsapp* o *Line* que han conseguido que los SMS queden en desuso.

La cuarta generación (4G) aumenta nuevamente las velocidades hasta los 100 Mbit/s en movimiento y 1 Gbit/s en reposo, lo cual permite la inclusión de nuevos servicios, muchos de ellos aún por aparecer y otros ya existentes como la recepción de televisión en Alta Definición.

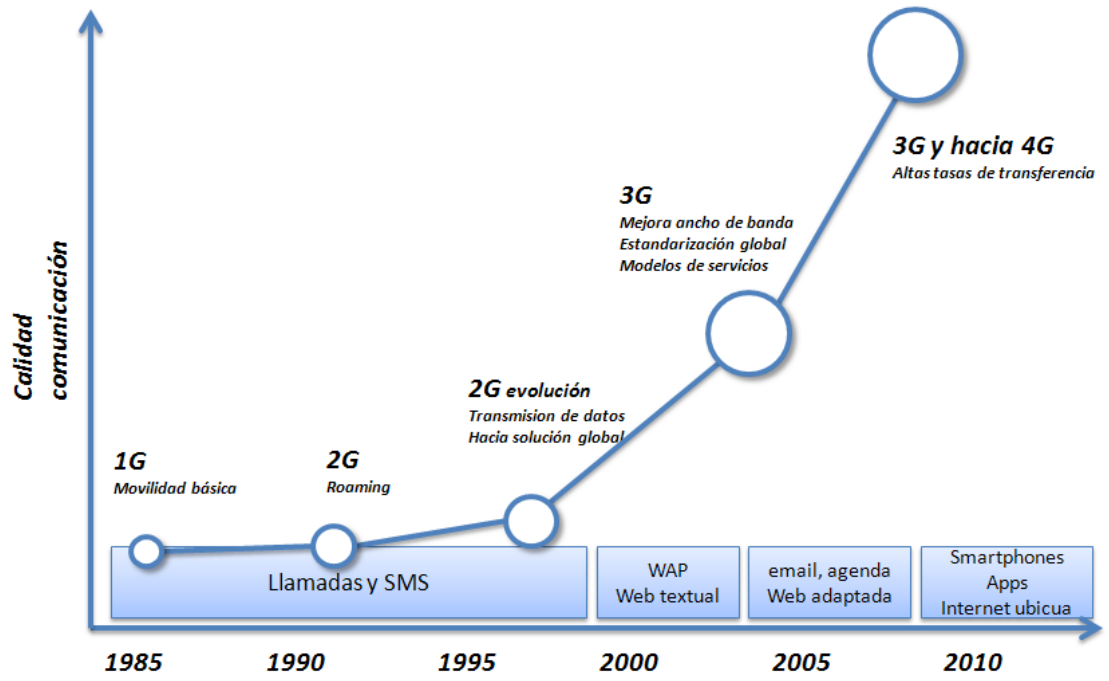


Figura 5. Evolución de la calidad de las comunicaciones en las distintas generaciones de telefonía móvil [8].

Actualmente, las empresas de telecomunicación se encuentran desarrollando la próxima generación (5G) la cual está previsto que esté lista para el año 2020. La compañía Ericsson ha realizado pruebas en las que ha conseguido alcanzar velocidades de 5 Gbit/s [9].

2.2 Sistemas Operativos y Aplicaciones Móviles

Como se ha mencionado en el apartado anterior, los *Smartphones* aparecieron en la tercera generación de teléfonos móviles. Aunque antes de la aparición de los *Smartphones* los teléfonos poseían SO, estos fueron los primeros dispositivos en poseer SO avanzados similares a los de los ordenadores personales. Sin embargo, los SO móviles son mucho más simples y están orientados fundamentalmente a la conectividad.

Hasta la aparición de los *Smartphones* predominaban los denominados sistemas propietarios, es decir, cada empresa de telefonía poseía su propio SO adecuado a las características del hardware de sus terminales.

Con los sistemas propietarios coexistían tres SO más avanzados: *Symbian* de *Nokia*, *Windows Mobile* y *PalmOS* de *Palm*. Los dos primeros fueron los primeros en ser instalados en diversos modelos de diferentes marcas.

En el año 2002 la compañía canadiense *RIM* saca *BlackBerry* que arrasaría en el sector corporativo por la inclusión del correo en el terminal móvil, esta característica luego sería tomada por los *Smartphones* [10].

Cinco años después, en el año 2007 *Apple* presenta el *iPhone* y por consiguiente la aparición de su SO *iOS*. Pese a que esta primera versión no era un *Smartphone*, *Apple* sentaría las bases del futuro de la telefonía móvil: pantalla táctil, acelerómetros para reaccionar a la posición del terminal, desaparición del teclado físico, gran pantalla y grandes capacidades multimedia.

En ese mismo año *Google* lanza la primera versión de su sistema *Android*, a diferencia de *iOS*, este SO surge de la alianza de varias compañías y es de código abierto. Desde ese año ambos SO se han ido replicando y mejorando, copando entre ambos prácticamente la totalidad del mercado. El claro ejemplo es que en el año 2008 Apple lanzó la primera tienda de aplicaciones, la denominada *AppStore* de *Apple* [11]. El día de su lanzamiento tenía solo 500 aplicaciones. Tal fue el éxito que siete días después se habían realizado 10 millones de descargas, y sólo una cuarta parte eran gratuitas. Pocos meses después *Google* lanzaría su propia tienda.

Actualmente ambos SO son los más utilizados como viene reflejado en la Figura 1 aunque también existen otros como *Windows Phone* y *BlackBerry OS* mucho menos utilizados.

Las diferencias más significativas entre *iOS* de *Apple* y *Android* de *Google* son:

- El software libre y abierto de *Google* frente a la licencia propietaria de *Apple*.
- La variedad de dispositivos a los que llega *Android* frente a la exclusividad de *iOS* en el *iPhone*.

Ambos SO poseen una gran cantidad de aplicaciones en su tienda que no para de aumentar, frente a sus competidoras que poseen un número mucho menor.

En la siguiente figura se pueden observar las principales diferencias entre SO:

	Apple iOS 6	Android 4.2	Windows Phone 7	BlackBerry OS 7	Symbian 9.5
Compañía	Apple	Open Handset Alliance	Windows	RIM	Symbian Foundation
Núcleo del SO	Mac OS X	Linux	Windows CE	Mobile OS	Mobile OS
Familia CPU soportada	ARM	ARM, MIPS, Power, x86	ARM	ARM	ARM
Lenguaje de programación	Objective-C, C++	Java, C++	C#, muchos	Java	C++
Licencia de *software*	propietaria	*software* libre y abierto	Propietaria	propietaria	*software* libre
Año de lanzamiento	2007	2008	2010	2003	1997
Motor del navegador web	WebKit	WebKit	Pocket Internet Explorer	WebKit	WebKit
Soporte Flash	No	Sí	No	Sí	Sí
HTML5	Sí	Sí	Sí	Sí	No
Tienda de aplicaciones	App Store	Google Play	Windows Marketplace	BlackBerry App World	Ovi Store
Número de aplicaciones	400.000	300.000	50.000	30.000	50.000
Coste publicar	$99 / año	$25 una vez	$99 / año	sin coste	$1 una vez
Plataforma de desarrollo	Mac	Windows, Mac, Linux	Windows	Windows, Mac	Windows, Mac, Linux
Actualizaciones automáticas del S.O.	Sí	depende del fabricante	depende del fabricante	Sí	Sí
Soporte memoria externa	No	Sí	No	Sí	Sí
Fabricante único	Sí	No	No	Sí	No
Variedad de dispositivos	modelo único	muy alta	baja	baja	muy alta
Tipo de pantalla	capacitativa	capacitiva /resistiva	capacitativa	/resistiva capacitativa	capacitiva /resistiva
Aplicaciones nativas	Sí	Sí	No	No	Sí

Figura 6. Comparativa entre sistemas operativos móviles [12].

Volviendo a las aplicaciones, recientemente han aparecido *apps* (nombre utilizado coloquialmente) para una infinidad de usos que explotan todo el potencial de los *Smartphones*. Además de la extensa variedad de juegos, que utilizan el potencial gráfico de estos teléfonos, han aparecido muchas aplicaciones de redes sociales como *Whatsapp*, *Facebook* o *Twitter*. A las que se han sumado GPS sociales como *Waze* o *Foursquare*, otras controvertidas como *Uber* que aspiran a competir con los servicios de taxi tradicionales, aplicaciones deportivas como *endomondo* o *Runtastic* que permiten compartir tus progresos en las redes.

También han aparecido aplicaciones multimedia que se han sumado a la conocida *Youtube*, además de que todas las webs de contenido multimedia poseen su aplicación, prácticamente todas las cadenas de televisión han lanzado la suya para poder visualizar contenidos de la cadena o incluso interactuar a tiempo real con la programación.

En otro apartado se encuentran aplicaciones más sencillas que son muy descargadas como simuladores de instrumentos, linternas que usan el flash de la cámara para iluminar, mandos a distancia de un televisor, grabadoras de voz, etc.

En definitiva, casi cualquier ente u organización tiene su aplicación o tiene intención de crearla en un futuro cercano. Tener una aplicación se ha convertido tan importante como tener una página web. Aunque actualmente parezca que sea un mercado muy explotado y sea difícil sorprender siguen apareciendo nuevas aplicaciones ya que el potencial es muy grande, además los elementos hardware de los nuevos dispositivos (nuevos sensores, mejores cámaras, mejores procesadores) precisan de la aparición de nuevas aplicaciones capaces de utilizar todo el potencial que poseerán las futuras generaciones de *Smartphones*.

2.3 Historia del Registro y el Procesamiento del Sonido

El sonido que es capaz de reconocer un ser humano está compuesto de ondas elásticas que se producen cuando un cuerpo vibra y genera oscilaciones de la presión del aire, estas son convertidas en ondas mecánicas en el oído humano y percibidas por el cerebro [13]. La propagación del sonido involucra transporte de energía sin transporte de materia.

Sus magnitudes físicas son las de cualquier onda: Longitud de onda, frecuencia, amplitud y fase. Aunque para caracterizar un sonido complejo se analiza:

- La Potencia Acústica: la cantidad de energía radiada al medio.
- El espectro de frecuencias: la distribución de la potencia acústica entre las diversas ondas que componen el sonido.

La historia del registro de sonidos se remonta al año 1857 cuando el inventor francés Leon Scott patenta el fonoautógrafo [14]. Este dispositivo fue el primer aparato capaz de transcribir un sonido de forma artificial a un medio visible, pero no se podía reproducir posteriormente.

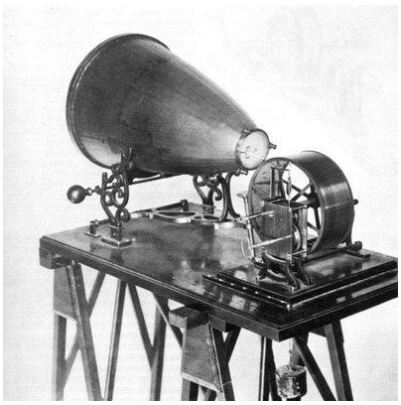


Figura 7. Fonoautografo creado por Leon Scott [14].

El fonoautografo estaba considerado como un elemento de laboratorio para el estudio de la acústica ya que gracias a él se podía estudiar el sonido y el habla, además de la frecuencia de un tono musical.

Dos décadas después, en 1877, Thomas Edison, basadose en los principios del fonoautografo crea el fonógrafo, el primer artefacto capaz de grabar y reproducir sonidos. El fonógrafo utiliza un sistema de grabación mecánica en el cual las ondas sonoras son transformadas en vibraciones y estás quedan registradas en un surco trazado de forma vertical en un cilindro. Para reproducir el sonido, una aguja pasa por un surco, recoge las vibraciones y las amplifica.

Figura 8. Fonógrafo creado por Thomas Edison [14].

Tan sólo 10 años después, el alemán Emile Berliner inventa en 1887 el gramófono, predecesor del tocadiscos. Este dispositivo remplaza el cilindro vertical por un plato giratorio horizontal que giraba a 80 RPM aproximadamente gracias a un motor a cuerda que después sería sustituido por un motor síncrono.

Figura 9. Gramófono creado por Emile Berliner [15].

El gramófono de Berliner acabó imponiéndose al fonógrafo de Edison por el menor coste de producción de los discos, con un único molde se podían realizar miles de copias mientras para grabar un cilindro de fonógrafos se necesitaban varios fonógrafos. Además el mecanismo del gramófono era más barato, sencillo y robusto.

Desde la aparición del fonógrafo se había estado investigando con el registro magnético, en 1911 con la invención del Tubo de Audión por Lee DeForest se conseguía amplificar los campos magnéticos, lo que propició la aparición del magnetofón de alambre. Pero no fue hasta el año 1930 cuando ya se consiguió grabar con suficiente calidad y lanzar al mercado. Fue muy utilizado por los aliados en la Segunda Guerra Mundial.

Figura 10. Magnetofón de alambre [14].

En 1925 aparecieron los tocadiscos gracias a la invención de las válvulas termoiónicas que permitían tener el control del volumen en reproducción. El tocadiscos está constituido por un plato al igual que el gramófono, en cambio rota más lento, a 33 RPM, y por tracción eléctrica además de poseer una púa que recorre el surco del disco.

En 1963 la empresa *Philips* introduce el casete compacto con idea de reducir el tamaño de los magnetófonos y de sus cintas. El casete estaba compuesto de una caja plástica cerrada que contenía un carrete de unos 100 metros de cinta magnética. La reducción del ancho de la cinta y la velocidad de reproducción produjo que el casete perdiera calidad respecto al magnetófono. El casete compacto sirvió de inspiración de otros soportes como el *VHS*, el *Casete Compacto Digital* o el *mini DV* utilizado en videocámaras.

Después llegaría la revolución digital con la reducción significativa del tamaño del soporte en comparación con los antiguos sistemas analógicos. Los nuevos dispositivos digitales tienen unos costes de grabación mucho más bajos y su vida útil es mucho mayor.

Sony modernizó el casete compacto en 1980, realizando la grabación en la cinta de manera digital consiguiendo una mejora de calidad y disminución de tamaño de la longitud de la cinta. Este casete sería muy popular, rivalizando durante años con el *CD* como soporte de audio digital.

El primer dispositivo de almacenamiento digital fue el *Laserdisc* lanzado en el 1978. El *Laserdisc* no tuvo mucho éxito debido a que apareció poco antes que las cintas *VHS*. Aunque no gozara de mucho éxito fue predecesor de los *CD*. El *CD*, que utiliza la misma tecnología que el *LaserDisc* pero con un soporte de menores dimensiones, fue lanzado en 1980. El sistema óptico lo desarrolló *Philips* mientras que la lectura y codificación digital fue desarrollada por *Sony*. Ambos desarrolladores normalizaron la duración en 70 minutos con el objetivo de poder grabar la *Novena Sinfonía de Beethoven* en un único disco. El *CD* consiguió una gran popularidad, desde entonces se han vendido más de 200 mil millones de copias.

Figura 11. Un Laserdisc (izquierda) comparado con un CD (derecha) [14].

Años después, a comienzo del siglo XXI, aparecería una segunda generación de soportes digitales como el *Blu-Ray* y *DVD*, una evolución del *CD*, y nuevos formatos como el MP3 y WAV.

La tecnología MP3 fue desarrollada en Alemania por tres científicos del instituto tecnológico de Fraunhofer. Pronto el formato MP3 se convirtió en un formato de audio con compresión de alta calidad ocupando hasta 15 veces menos que el equivalente sin compresión. Aunque posee una ligera pérdida de calidad en equipos de alta fidelidad, su reducido tamaño le posibilita ser el formato de audio más utilizado en internet. Dada la popularidad que ha alcanzado se crearon memorias flash y reproductores MP3 para su reproducción fuera de un ordenador y los nuevos *Smartphones* son capaces de reproducirlo.

Aunque el MP3 sea el formato más extendido, dado que utiliza una compresión con pérdidas no sirve para realizar un procesado de audio. Para realizar un procesado es recomendable utilizar formatos sin perdidas. Estos formatos tienen como parámetros principales:

- **Frecuencia de muestreo:** Viene determinada por el Teorema de Muestreo de Nyquist puesto que el sistema auditivo humano no es capaz de percibir frecuencias superiores a 20kHz no tiene sentido utilizar frecuencias de muestreo mucho más altas que 44kHz.
- **Número de bits por muestra**. Determina la precisión con la que se reproduce la señal original y el rango dinámico de la misma. Se suelen utilizar 8 (para un rango dinámico de hasta 45 dB), 16 (para un rango dinámico de hasta 90 dB) o 24 bits por muestra (para 109 a 120 dB de rango dinámico). El más común es 16 bits.
- **Tipo de compresión *(si la hay)*:** Compresión con pérdidas (*lossy*) y sin pérdidas (*lossless*).
- **Tasa de bits:** determina el número de bits de información necesarios por unidad de tiempo.

Entre los formatos sin perdidas el más utilizado es el WAV propiedad de *Microsoft* e *IBM*. Aunque el formato sea compatible con casi cualquier códec, se utiliza principalmente con el formato PCM (no comprimido) y al no tener pérdidas de calidad es el más adecuado para el procesado de audio y el uso profesional. Para conseguir un formato WAV con calidad CD audio será necesario que el sonido se grabe a una frecuencia de muestreo 44 KHz y con 16 bits por muestra. Las dos grandes limitaciones de este formato son:

- Es un formato pesado, cada minuto de grabación ocupa en torno a 10 megabytes.
- Solo se pueden grabar ficheros de 4GB como máximo.

También existen otros formatos sin pérdida pero con compresión, lo cual los hace más ligeros, como *FLAC* y *Apple Lossless*. Estos formatos son muy interesantes para ser reproducidos en equipos de alta fidelidad pero al poseer una compresión resulta dificultoso realizar un procesado de ellos.

El procesado de audio comienza con la invención de las válvulas termoiónicas que poseían los tocadiscos. Gracias a ellas se empezó a controlar el volumen de la reproducción y la tonalidad del sonido (control de graves y agudos).

Aunque es años más tarde, con la revolución digital, cuando el procesado de señales y el procesado de audio (el sonido no deja de ser una señal) comenzó a desarrollarse de una manera más notable.

Al digitalizar las señales se consigue mejorar la transmisión, almacenaje y manipulación de una señal. Es menos sensible a las interferencias e inmune al ruido y con el tiempo no se degrada.

Para realizar un procesado digital de una señal, si se tiene como entrada una señal analógica, es necesaria una conversión Analógico/Digital que normalmente se realizará con la ayuda de un microprocesador. Para ello se toma la señal analógica y se recogen muestras con una frecuencia de muestreo determinada por el *Teorema de Muestro de Nyquist*. Al realizar esta conversión siempre habrá una pérdida de información entre la señal analógica y la señal digital obtenida.

Una vez que se tiene la señal digital, las transformaciones de señales son sencillas de realizar. Entre todas las transformaciones destaca las que realiza la transformada de Fourier. Esta transformada convierte la señal del dominio del tiempo al dominio de la frecuencia lo cual facilita la eliminación de ruido y la aplicación de filtros: Filtro Paso Bajo, Filtro Paso Alto, Filtro Paso Banda, etc.

2.4 Ayudas Técnicas a la Sordera

Actualmente existe un amplio abanico de ayudas a las personas con disfunciones auditivas. Este tipo de ayudas ya no se limitan a los vetustos audífonos, que además han evolucionado sustancialmente, sino que también han aparecido nuevas soluciones. A continuación se detallarán las más relevantes:

2.4.1 Audífonos

Los audífonos llegaron al mercado a principios del siglo XX, eran pesados y nada adecuados para llevarlos encima. Los primeros audífonos consistían normalmente en un micrófono independiente, un amplificador, auriculares y una voluminosa batería [16]. El dispositivo funcionaba mejor cuando se colocaba en la mesa y se utilizaba con un par de auriculares. Aunque la batería era grande, tan sólo duraba un par de horas. Su precio era muy elevado.

La invención del transistor en 1947 revolucionó la tecnología de los audífonos lo cual hizo posible fabricar audífonos mejores y más pequeños. Como la capacidad de amplificación de los transistores era mucho mayor a las antiguas válvulas, se conseguía una mejor calidad, y sumado a la aparición de nuevos tipos de baterías como las pilas, se lograba una mayor autonomía.

Figura 12. Audífono con tecnología de transistores [17].

En los años 60 se logró disminuir el tamaño lo suficiente como para que ya se pudieran colocar detrás de la oreja.

Pero fue a mediados de los 80 cuando la revolución digital logro mejoras espectaculares en la efectividad respecto a los anteriores audífonos analógicos. Esta nueva generación de audífonos se componen de un pequeño ordenador programable que es capaz de amplificar millones de diferente señales sonoras con mucha precisión. Esta tecnología ha ido evolucionando y desde finales de los años 90 se ha establecido, logrando llegar a un gran número de personas.

Figura 13. Diferentes modelos de Audífonos que hay en el mercado en la actualidad [18].

Los nuevos audífonos digitales mejoran la capacidad de audición de las personas con discapacidad auditiva hasta tal punto que incluso personas con pérdidas profundas de audición pueden desarrollar una vida casi normal.

2.4.2 Otras Ayudas Técnicas

Además de los audífonos existen más ayudas técnicas para personas con problemas de sordera [19]:

- **Estimuladores vibrotáctiles**

  Son aparatos que recogen el sonido, especialmente el correspondiente al habla, convirtiéndolo finalmente en vibraciones cuya naturaleza depende del tipo de sonido captado. En este apartado estaría englobado el ejemplo *PocketVib cordless vibrator* puesto en el apartado de motivación 1.1 del presente documento.

- **Sistemas FM**

  Son sistemas que transmiten la señal sonora mediante ondas de alta frecuencia a un receptor que se conecta por entrada directa de audio o por inducción magnética al audífono. También se utilizan en conferencias o clases permitiendo al usuario de audífonos evitar las interferencias.

- **Sistemas de reconocimiento de habla**

  Son programas de ordenador que, gracias a un micrófono, pueden identificar los fonemas y trascribirlos sobre la pantalla del ordenador como caracteres escritos. El programa más conocido es *Viavoice* desarrollado por *IBM*.

- **Amplificador Magnético Portátil**

  Se trata de un pequeño amplificador que envía la conversación telefónica directamente al audífono sin interferencias de ruido de fondo. El amplificador se coloca en el auricular del teléfono.

- **Dispositivo Luminoso por Destellos**

  Este es un dispositivo luminoso que luce intermitente con la recepción de una llamada telefónica, la alarma del despertador u otro tipo de alarma.

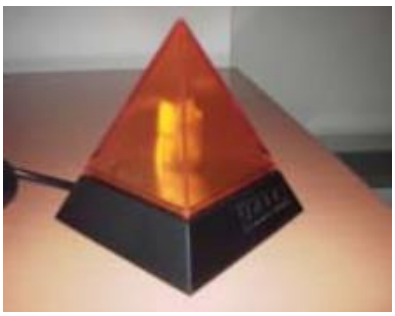


Figura 14. Dispositivo Luminoso por Destellos [20].

- **Aviso de Alarma para Despertador**

  Este aparato avisa de la alarma de cualquier despertador mediante un potente vibrador situado debajo de la almohada o por luz.

3 Entorno de Desarrollo

En este apartado se van a describir las herramientas y tecnologías que han sido más importantes para el desarrollo de la aplicación objetivo del presente TFG. Estas herramientas son: el soporte al desarrollo que brinda el SO *Android*, el *IDE Eclipse*, y su vinculación con *Android*, y la librería *JTransforms* utilizada para realizar una *FFT (Transformada Rápida de Fourier)*.

3.1 Android

3.1.1 Breve Introducción Histórica

La historia de *Android* se remonta al año 2003, cuando se funda la firma *Android Inc* [21]. En sus inicios esta pequeña compañía se dedicaba a desarrollar software para dispositivos móviles. En el año 2005, *Google* compra *Android Inc* y durante un tiempo, lleva con gran hermetismo las actividades de esta compañía.

Hasta que en el año 2007 anuncia la creación de un nuevo SO para dispositivos móviles, construido sobre la versión 2.6 del kernel de *Linux* y de código abierto. Al mismo tiempo también anuncia la creación de la *Open Handset Alliance*, un consorcio de 78 compañías de hardware, software y telecomunicaciones dedicadas al desarrollo de estándares abiertos para dispositivos móviles.

La primera versión de *Android* fue lanzada en septiembre del año 2008 con el nombre de Apple Pie (Tarta de Manzana). Un mes más tarde, se lanzaría al mercado el primer móvil con el novedoso sistema operativo: el *HTC Dream*.

El lanzamiento de *Android* revolucionó el mundo de la telefonía con la inclusión de un nuevo SO libre y de código abierto bajo licencia Apache, la cual permite a los desarrolladores la libertad distribuirlo, modificarlo y lanzar versiones modificadas de ese software siempre que se mencione la fuente. El objetivo de *Android* era crear estándares abiertos que llegarán al mayor número de terminales posible, terminales cuyas compañías *Google* mantiene un consorcio. Con el paso de los años se ha demostrado que aquella propuesta de *Google* ha llegado a ser un rotundo éxito.

3.1.2 Arquitectura de Android

Después de hacer una breve introducción histórica de *Android* y explicar porque fue tan interesante su aparición en el mercado, se procede a detallar su estructura. Esta estructura se puede dividir en capas.

Como ya se ha mencionado *Android* utiliza como base el kernel de *Linux* en su versión 2.6. El kernel es el núcleo fundamental de un SO y se define como la parte que se ejecuta en modo privilegiado. Es el principal responsable de facilitar a los distintos programas el acceso básico al hardware de un dispositivo y el encargado de gestionar recursos, a través de servicios del sistema [22] .

Aunque *Android* use de base el kernel de *Linux*, este no es idéntico al de *Linux* ya que incorpora nuevos mecanismos necesarios para el manejo del hardware de dispositivos móviles como servicios de seguridad, el manejo de la memoria, el multiproceso, la pila de protocolos y el soporte de drivers para dispositivos. Esta es la única capa que es dependiente del hardware.

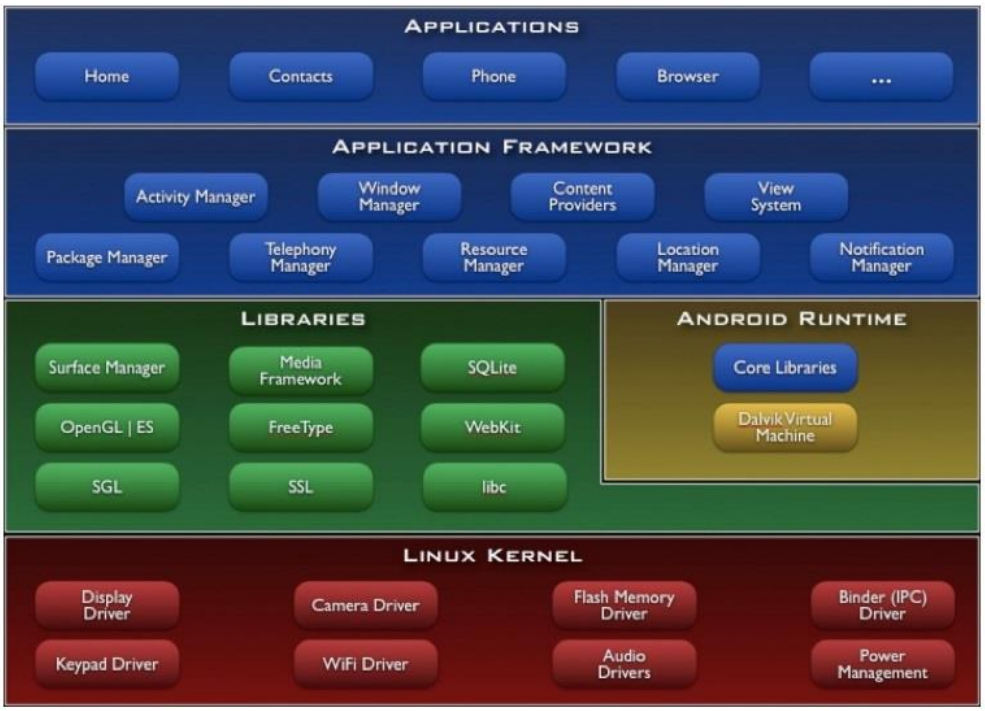


Figura 15. Esquema de la Arquitectura de Android [23].

El *Android Runtime* es una capa basada en el concepto de máquina virtual de *Java*. Debido a las limitaciones de los dispositivos donde se ejecuta *Android*, no fue posible utilizar la máquina virtual de Java estándar [23]. Google tomó la decisión de crear una nueva más liviana y optimizada para dispositivos móviles: la *máquina virtual Davlik*.

La siguiente capa son las librerías nativas desarrolladas en *C/C++*. Son usadas en componentes de *Android* como el navegador, la reproducción de material multimedia, el motor de gráficos 2D y 3D, bases de datos o los servicios de encriptación. Muchas de estas librerías utilizan proyectos de código abierto.

En una capa más superior, se encuentra el denominado *Applicatton Framework* que es utilizado para simplificar la reutilización de componentes. Las aplicaciones pueden publicar sus capacidades y otras interactuar con ellas, logran una optimización del sistema completo.

En el nivel más alto se encuentran el conjunto de las aplicaciones instaladas en el dispositivo. Todas las aplicaciones deben de ser ejecutadas en la *máquina virtual Davlik* para garantizar la seguridad del sistema.

Normalmente las aplicaciones *Android* se desarrollan en lenguaje *Java* utilizando el *Android SDK* que incluye un conjunto de herramientas de desarrollo.

Aunque también existen otras alternativas como desarrollar aplicaciones en código nativo *C/C++*, para el cual existe un kit de desarrollo denominado *Android NDK*.

3.1.3 Desarrollo Basado en Java

Como se acaba de mencionar, si el programador decide desarrollar *Android* en *Java*, deberá utilizar el SDK de *Android*. El SDK (*Software Development Kit*) que comprende de un depurador de código, biblioteca, un simulador de teléfono, documentación, ejemplos de código y tutoriales [24].

Android no utiliza los estándares establecidos de *Java*, lo cual es un problema para el desarrollo si se quieren importar aplicaciones *Java* escritas para otras plataformas, ya que la compatibilidad es escasa con ellas.

Android incorpora de *Java* su sintaxis y semántica. También utiliza un gran número de biblioteca de clases y API *(Application Programming Interface*) aunque hay otros muchos que no, como todos los paquetes de clase *java.x*.

Por tanto, existe cierta compatibilidad entre *Java* y *Android* aunque a la hora de hacer desarrollos será necesario conocer estas limitaciones. Si se quieren incorporar componentes externos desarrollados en *Java* a un proyecto *Android*, será posible siempre y cuando este nuevo componente utilice las librerías y paquetes que *Java* comparte con *Android*.

3.2 Eclipse

Eclipse es un *Entorno de Desarrollo Integrado* (*IDE*) creado originalmente por *IBM* en el año 2001 [25] y respaldado por un consorcio de vendedores de software. Dos años más tarde, en el año 2004, se crea la *Fundación Eclipse* como una corporación independiente sin ánimo de lucro cuyo objetivo es el de actuar como administrador de la comunidad *Eclipse* y asegurar el desarrollo de *Eclipse* en código abierto.

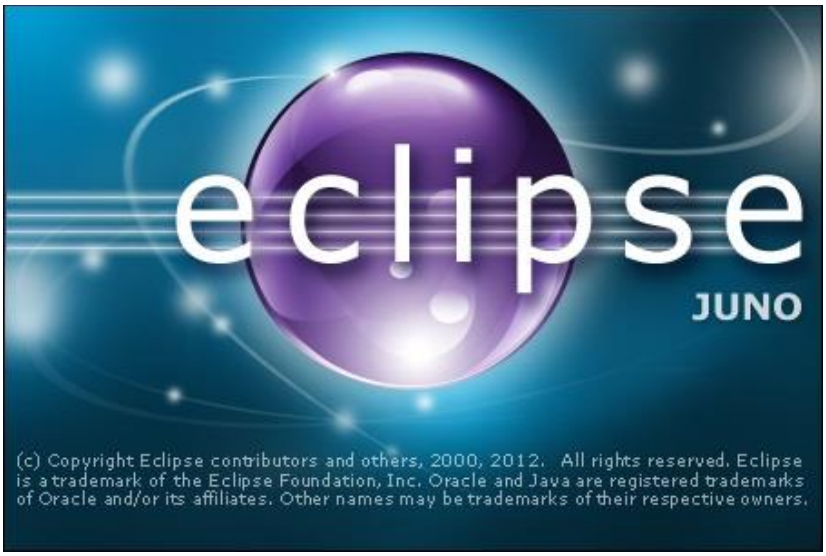


Figura 16. Cabecera de inicio del IDE Eclipse en su versión Juno.

Los IDE son programas informáticos compuestos por un conjunto de herramientas de programación. Pueden dedicarse en exclusiva a un solo lenguaje de programación o bien pueden utilizarse para varios [26]. La mayoría de los IDE poseen, al menos los siguientes componentes:

- Un editor de texto.
- Un compilador.
- Un intérprete.
- Un depurador.
- Un cliente.
- Posibilidad de ofrecer un sistema de control de versiones.
- Factibilidad para ayuda en la construcción de interfaces gráficas de usuario.

Visual Studio, *NetBeans* y el mencionado *Eclipse* son los IDE más utilizados hoy en día.

Visual Studio desarrollado por Microsoft soporta la mayoría de lenguajes pero solo corre en ordenadores con SO de *Microsoft*.

NetBeans, en cambio es de código abierto y se utiliza fundamentalmente para el desarrollo en *Java*, aunque existen extensiones para realizar otros tipos de desarrollos. Cabe destacar que existe una extensión para desarrollar en *Android*, aunque no es muy utilizada.

Eclipse es un *IDE* multiplataforma, de código abierto proporciona soporte para la mayoría de lenguajes de programación, como *C, C++, JavaScript, Fortran, PHP o Python*, pero su uso más extendido es con *Java*. Uno de sus potenciales erradica en la utilización de módulos (en

inglés denominados *plugins*) que agregan las funcionalidades y lenguajes que requieran las necesidades de cada programador.

Por todos estos motivos se ha decidido utilizar *Eclipse* con el *plugin* ADT *(Android Development Tools)*, que permite el uso de las herramientas necesarias para el desarrollo de proyectos de aplicaciones *Android*.

Cabe destacar otra alternativa para el desarrollo en *Android*, llamada *Android Studio*, que ha aparecido recientemente promovida por *Google*. Actualmente se encuentra en versión *Beta*, pero con nuevas ayudas al programador de las que carece el *plugin* ADT de *Eclipse* y un gran soporte de *Google*, lo cual promete tener un gran potencial para futuros desarrollos de aplicaciones *Android*.

3.3 Librería JTransforms

JTransforms [27] es una librería de *Java* de código abierto que es capaz de realiza las siguientes transformaciones: *Discrete Fourier Transform* (DFT), *Discrete Cosine Transform* (DCT), *Discrete Sine Transform* (DST) y *Discrete Hartley Transform* (DHT). Al distribuirse en un archivo *.jar* y no depender de clases *java.x* es totalmente válida para *Android*.

Se trata de una librería escrita únicamente en *Java* lo cual consigue tener un buen rendimiento a la hora de realizar las transformaciones. A esto habría que añadirle, que a diferencia de otras, es capaz de realizar transformaciones *FFT* con entradas únicamente reales, lo cual mejora aún más el rendimiento.

JTransforms también destaca por ser sencilla de implementar, dado que acepta como entrada un *array* cuya longitud sea variable. Asimismo provee una biblioteca de métodos que destacan por la sencillez de implementación.

Se distribuye bajo licencia BSD2, es decir, se trata de una licencia de software libre permisiva.

4 Diseño de la Aplicación

En el presente apartado se va presentar los requisitos que deberá cumplir la aplicación desarrollada en este TFG. Se mostrarán las diferentes alternativas para la consecución de los mismos indicando cuáles son las ventajas e inconvenientes de cada solución propuesta. Posteriormente se describirá la solución elegida con ayuda de diagramas de flujo.

4.1 Planteamiento

El propósito de este trabajo es realizar una aplicación que reconozca un número determinado de sonidos *familiares* que aparecen en el entorno del hogar, cuando la aplicación detecte uno de estos sonidos aparecerá un mensaje luminoso en la pantalla del dispositivo. El objetivo es que este desarrollo sea de utilidad para personas con problemas de audición a la hora de identificar determinados sonidos cuando se encuentren en su hogar.

4.2 Diseño de la Solución

Para conseguir esto, será necesario grabar sonidos con el uso del micrófono del teléfono, posteriormente se realizaran diferentes operaciones sobre este sonido para extraer unas determinadas características. Finalmente se realizará una comparación entre sonidos.

El diseño de la solución del problema se obtendrá tras un estudio exhaustivo de todas las alternativas planteadas. Se buscará la solución más óptima y la que mejor se adapte a los requisitos primando siempre la sencillez en el diseño.

4.2.1 Versión de la Aplicación

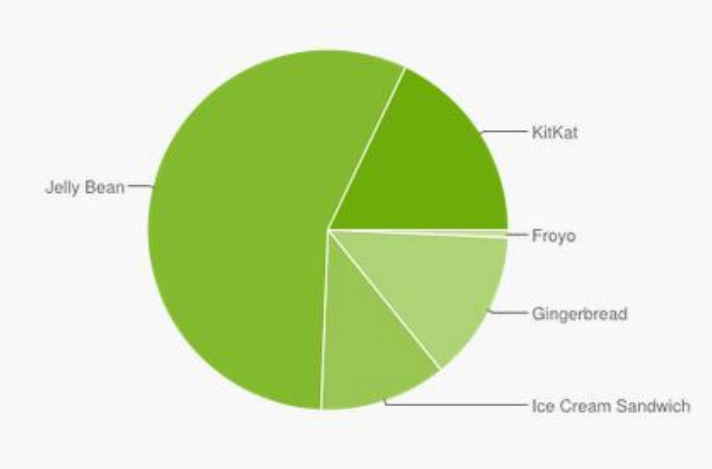
El primer requisito a fijar de la aplicación es la versión de *Android* que se quiere utilizar. Esto definirá el número de usuarios a los que se quiere llegar como potenciales clientes. Una versión muy reciente poseerá las más novedosas características del SO pero tendrá el contrapunto de que no todos los usuarios podrán usarla; solo aquellos usuarios que tengan instaladas las versiones recientes. En cambio, desarrollando en una versión más antigua que llegue a la totalidad de los usuarios, esta no poseerá determinadas características que pueden ser necesarias para determinadas aplicaciones. El desarrollador debe buscar un equilibrio entre la funcionalidad y llegar al máximo número de usuarios posibles.

A continuación se muestran las versiones de *Android* existentes hasta la fecha y sus características más importantes [28]:

- Android 1.0 *Apple Pie* .Lanzada en septiembre de 2008 es la primera versión comercial del software. Ya posee elementos como: Correo Push, Google Maps, Android Market, Navegador, YouTube, barra de notificación, Wi-Fi, Bluetooth, reproducción multimedia, etc.
- Android 1.1 *Banana Bread.* Lanzada en febrero de 2009 para fundamentalmente solventar problemas de la primera versión.
- Android 1.5, *Cupcake*: Lanzada en abril de 2009, incluye entre sus funciones un teclado QWERTY virtual, widgets, auto-rotación de pantalla, captura de vídeo o copiar y pegar.
- Android 1.6, *Donut*: Lanzada en septiembre de 2009. Incluye navegación mejora en el interfaz de búsqueda e incluye búsquedas por voz.
- Android 2.0, *Eclair*: Lanzada en diciembre de 2009. Nuevas características para la cámara como inclusión de zoom, optimización de la velocidad del hardware y GUI, y un nuevo navegador de internet con soporte de vídeo y HTML5
- Android 2.2, *Froyo*: Lanzada en mayo de 2010. Nuevas mejoras en rendimiento y memoria. Aparece el soporte a Flash.
- Android 2.3, *Gingerbread*: Lanzada en diciembre de 2010. Incluye mejoras de la gestión de la energía, del teclado virtual y soporte para NFC (Near Field Communication).Aparece soporte nativo a telefonía VoIP.
- Android 3.0-3.4, *Honey Comb*: Lanzada en mayo de 2011, son versiones de *Android* especialmente diseñada para tablets que con la siguiente actualización quedaron prácticamente inutilizadas.
- Android 4.0, *Ice Cream Sandwich*: Lanzada a finales de 2011 para todo tipo de dispositivos móviles (*SmartPhones* y *tablets*). Incluye pantalla principal con imágenes 3D, grabación de video a 1080p, *Android Beam* apoyándose en NFC, Wi-Fi Direct, nuevo navegador *Chrome,* etc. También proporciona soporte para teclados y mandos USB.
- Android 4.1-4.3, *Jelly Bean*: Lanzada en julio de 2012. Mejora el rendimiento de los menús, se renueva el motor gráfico, nuevo Bluetooth de baja energía.
- Android 4.4 *Kit-Kat.* Lanzada en enero de 2014.Integración de la cámara con la galería. Nuevos widget de ajustes rápidos. Novedades en el interfaz. Arreglos de bugs en versiones anteriores.

Como curiosidad el lector podrá haber percibido que hasta ahora, todas las versiones llevan nombre de dulces en inglés.

Por la actividad que realiza, la aplicación que se va a desarrollar no necesitará las novedades de las últimas versiones. Por lo cual no habrá limitaciones al respecto para que la aplicación tenga la máxima difusión posible.



Version	Codename	API	Distribution
2.2	Froyo	8	0.7%
2.3.3 - 2.3.7	Gingerbread	10	13.5%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	11.4%
4.1.x	Jelly Bean	16	27.8%
4.2.x		17	19.7%
4.3		18	9.0%
4.4	KitKat	19	17.9%

Figura 17. Cuota de mercado Android por versiones (Julio de 2014) [29]

Como se puede observar de la Figura 10 que representa la cuota de mercado *Android* por versiones, en la actualidad la mayor parte de los usuarios utiliza las últimas versiones *Jelly Bean* y *KitKat*. Estas versiones han tomado el relevo de *Gingerbread* que hasta hace poco era la más utilizada (en Enero del año 2014 poseía el 21% de la cuota).

Aunque la utilización de *Gingerbread* esté decreciendo drásticamente debido a la antigüedad de la versión (data del año 2010) sigue representando un 13.5% y dado que se quiere llegar al máximo número de dispositivos y no hay ninguna otra limitación, es recomendable incluir esta versión en el desarrollo. La versión *Froyo* posee un 0.7% y dado que las mejoras de *Gingerbread* respecto a esta son significativas se ha decidido no incluirla.

Eligiendo *Gingerbread* como versión mínima se garantiza la compatibilidad con el 99.3% de usuarios en la actualidad, ya que la compatibilidad hacia nuevas versiones es total. Adicionalmente como *Froyo* es una versión en extinción, este porcentaje tenderá a acercarse al 100%.

Una vez fijado el número potencial de usuarios de la aplicación el siguiente paso será realizar una descripción del diseño general de la aplicación.

4.2.2 Diseño General de la Aplicación

La aplicación deberá detectar 5 sonidos: el timbre de una casa, el timbre de un portal, el microondas, el teléfono y el sonido de una nevera abierta. Cada sonido será analizado y se le observarán las frecuencias características para realizar su identificación.

Una vez se inicie la aplicación aparecerá un botón de "Inicio de detección", una vez pulsado, la aplicación comenzará a realizar las siguientes operaciones secuencialmente, estas operaciones serán detalladas en los siguientes apartados:

1. Realizará una grabación con el micrófono del dispositivo con una duración en torno a los 2,5 segundos.
2. Una vez terminada la grabación se eliminarán los silencios de la grabación para eliminar información innecesaria.
3. Si al aplicar la eliminación de silencios, la grabación es muy corta, se desprecia la muestra.
4. Después de eliminar silencios, se divide la grabación en ventanas solapadas.
5. En cada ventana se aplica una función de enventanado.
6. Cada ventana se transformará del dominio del tiempo al dominio de la frecuencia con la Transformada de Fourier.
7. Se extraerán las 3 frecuencias más altas de cada ventana con su correspondiente Potencia Acústica.
8. Una vez extraídas las frecuencias de todas las ventanas, las frecuencias se compararán con las de los 5 sonidos.
9. Si la identificación coincide con uno de los 5 sonidos, aparecerá por pantalla un símbolo del sonido identificado.
10. En caso de no detectar nada se vuelve al punto 1 y vuelve a repetir el proceso. Este proceso finalizará cuando se vuelva a pulsar el botón "Fin de detección".

El programa poseerá un modo de inicio y fin automático de detección, en el cual se podrá especificar, opcionalmente, la hora de inicio y la hora de final que el usuario quiere que haya detección de sonidos. Esta hora el usuario la programará previamente. El objetivo de esta función es lograr un ahorro de batería cuando el usuario no requiera la detección de sonidos.

Adicionalmente la aplicación poseerá un apartado de estadísticas en el que se mostrará el número de sonidos identificados de cada tipo y un registro o *log* de los sonidos identificados con la hora y el día.

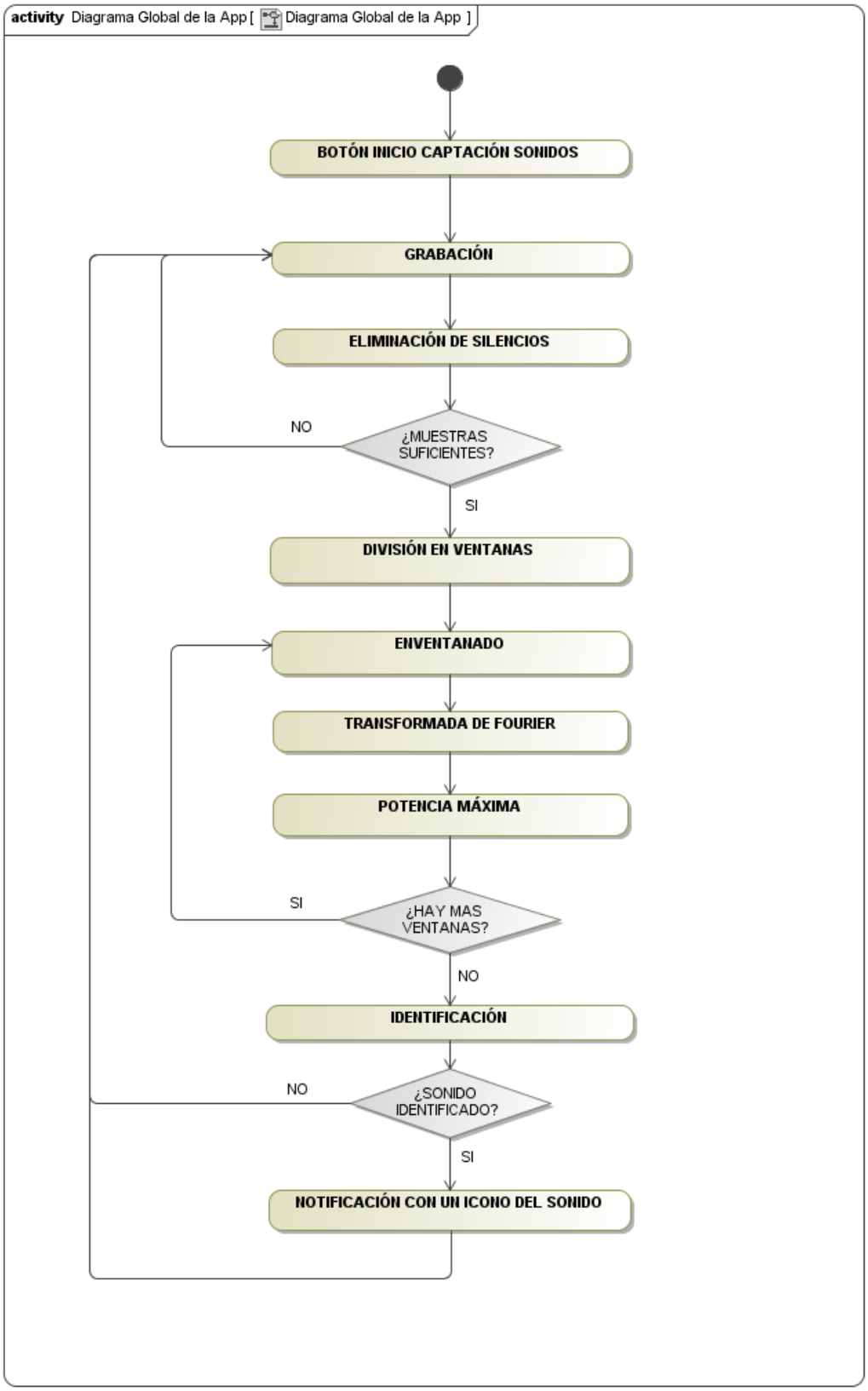


Figura 18. Diagrama global de actividad de la aplicación.

4.2.3 Diseño de la Grabación

La grabación se realiza con el micrófono del dispositivo. Dado que posteriormente se va a realizar un procesado, el formato en el que se grabe será sin compresión y sin pérdidas.

La onda que capta el micrófono del dispositivo *Android* necesita una amplificación y una conversión analógica-digital (ADC) para poder ser tratada posteriormente, ya que a la salida del micrófono existe una señal analógica con muy poca amplitud. El sistema *Android* realiza la amplificación y la conversión analógica-digital por sí sólo, por lo que el desarrollador solo tiene que realizar su desarrollo a partir de la señal digital que le ofrece *Android*.

Para que la señal capturada pueda ser procesada con comodidad posteriormente, la grabación debe tener las siguientes propiedades:

- La frecuencia de muestreo debe de ser de 22 KHz.
- La grabación debe de ser en un único canal: MONO.
- El número de bits por muestra debe ser de 16 bits.
- El tipo de codificación debe de ser PCM.

La frecuencia de muestreo viene determinada por el Teorema de Muestreo de Nyquist. El Teorema de Nyquist dice que para poder reconstruir una señal con exactitud es necesario que la frecuencia de muestreo sea superior al doble de la máxima frecuencia a muestrear [30]:

$$F_s > 2F_{MAX}$$

El espectro audible medio de un ser humano se encuentra entre las frecuencias de 20Hz y 20KHz aproximadamente.

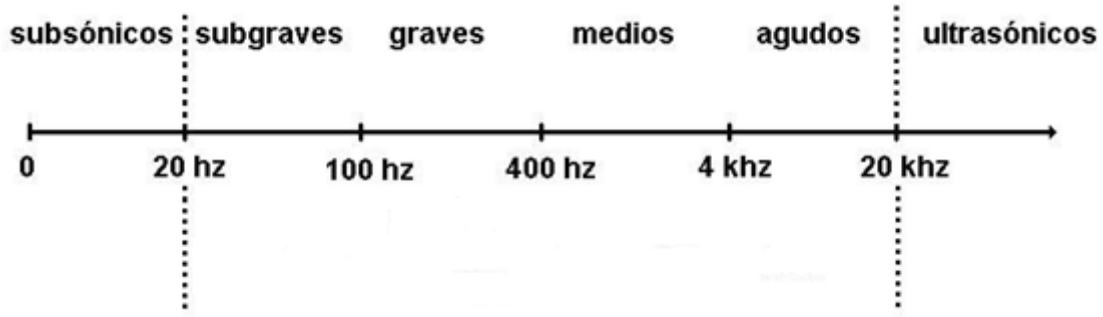


Figura 19. Rango de frecuencias audible por un ser humano [30].

Para cumplir el teorema de Nyquist y muestrear todo el espectro audible se suele utilizar una frecuencia de muestreo de 44 KHz, dicha frecuencia es la que utilizan las grabaciones y equipos de alta fidelidad. Pero como los sonidos que se van a capturar no superan frecuencias de 8 KHz, con la frecuencia de muestreo típica de 22 KHz es más que suficiente. Hay que tener en cuenta que muestreando a 44 KHz se obtendría una mejor resolución pero esto conllevaría multiplicar por dos el tamaño de los datos a procesar y, dado que estamos en un dispositivo móvil esto se vería afectado en el rendimiento de la aplicación.

La grabación se realiza en Mono, en un único canal. La adquisición en Stereo o Surround no aportaría información adicional para el procesado, por lo tanto con realizar la grabación en 1 canal será suficiente.

Como número de bits por muestra se ha elegido 16 bits. El número de bits por muestra corresponde al número de bits que utiliza el conversor ADC de *Android* para representar cada muestra analógica. Cuanto mayor sea el número de bits con mayor fidelidad se va representar la señal analógica. Para formatos sin compresión, *Android* ofrece codificación PCM para 8 y 16 bits. Con intención de mejorar el rendimiento se pensó en elegir 8 bits, ya que se necesitaba la mitad de datos para realizar el procesado que con 16 bits. Pero como para 8 bits la API no garantiza su funcionamiento [31], se realizaron pruebas en varios dispositivos y en ninguno funcionó, por lo cual se opta por codificación PCM y 16 bits como número de bits por muestra.

El resultado de la grabación queda almacenado en un *array* de bytes, sin cabecera, solo con la información de la señal.

4.2.4 Diseño de la Eliminación de Silencios

La eliminación de silencios parte de la señal digital obtenida en el módulo de grabación. La eliminación de silencios eliminará aquellos momentos de la grabación que carezcan de información útil para el procesado posterior. Estos *silencios* vendrán determinados por una comparación con el resto de la grabación. Después de la eliminación de silencios se espera que la grabación inicial de 2,5 segundos quede con una duración algo menor de 1,5 segundos.

El objetivo de este módulo es eliminar información inútil para el procesado posterior y aligerarlo, consiguiendo así, que todo el proceso de aquí en adelante sea más rápido, y la aplicación sea más eficiente en la utilización de recursos.

4.2.5 Diseño del Enventanado

La señal obtenida de la *eliminación de silencios* se divide en ventanas solapadas entre sí. Si se quisiera realizar una identificación del habla habría que seleccionar ventanas de tamaños del orden de 50-100 ms pero como se quieren captar sonidos podrán ser mayores para ganar exactitud, por lo cual con que sean de 400 ms será suficiente para la identificación de sonidos.

Las ventanas están solapadas por la mitad, es decir, cada 200 ms, de esta manera toda la información de la grabación tendrá la misma importancia. El número de ventanas responde a la siguiente ecuación experimental:

$$N_{VENTANAS} = \frac{DURACIÓN\ DE\ LA\ GRABACIÓN}{DURACIÓN\ MEDIA\ VENTANA} - 1$$

Como de la *eliminación de silencios* se espera que llegue una grabación de aproximadamente 1,5 segundos el número de ventanas esperado es:

$$N_{VENTANAS} = \frac{1{,}5\ segundos}{0{,}2\ segundos} - 1 = 6{,}5 \sim 6\ ventanas$$

En el resultado sale un número decimal, este se trunca, despreciando la información de la última ventana de la grabación, ya que no es deseable tener una ventana más pequeña que las demás. A parte de su tamaño, al ser la última ventana, su información no será tan interesante como las de sus antecesoras.

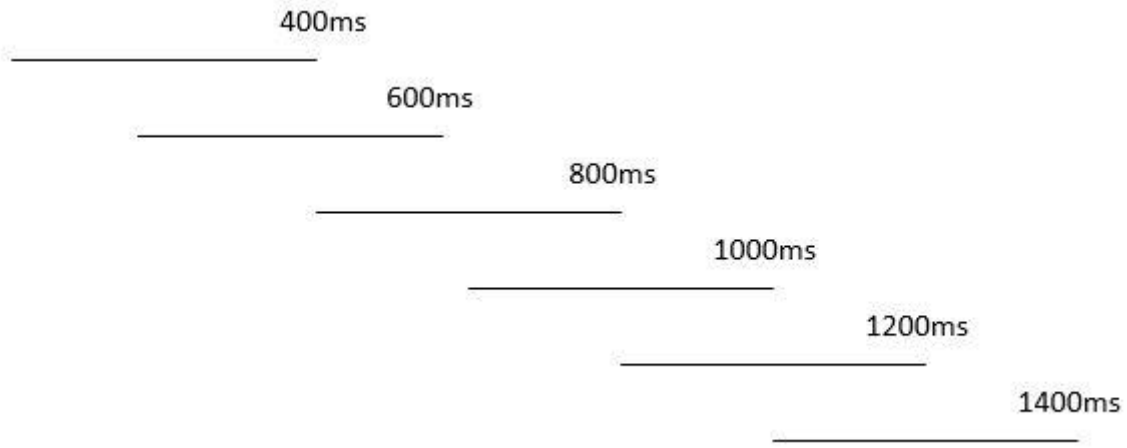


Figura 20. Representación de la distribución de las ventanas en una grabación.

Con un número de aproximadamente 6 ventanas será conveniente para realizar un procesado ya que lo deseable será que, al menos, la mitad de las ventanas den información útil.

Una vez dividida la toma en ventanas, será necesario realizar el enventanado o *windowing* antes de realizar la Transformada de Fourier.

El *windowing* o enventanado consiste en aplicar una función de ventana dentro de dicha ventana con el objetivo de prevalecer un rango de valores en el dominio de la frecuencia. Dentro de todas las ventanas que existen destacan:

- Ventana Rectangular
- Ventana Von Hann
- Ventana de Hamming
- Ventana Blackman

Se ha elegido la ventana de Hamming ya que da importancia a las frecuencias que aparecen en el centro de la ventana sin obviar del todo el resto de muestras. Se le da más importancia a las frecuencias del centro de la ventana debido a la manera que las ventanas están solapadas. Dado que están solapadas y hay información repetida en distintas ventanas, se toman los valores centrales que van a ser los más representativos.

La función de ventana de Hamming responde a la siguiente ecuación:

$$w(n) = \alpha - \beta \cos\left(\frac{2\pi n}{N-1}\right)$$

Dónde:

$$\alpha = 0{,}54\,, \beta = 0{,}46$$

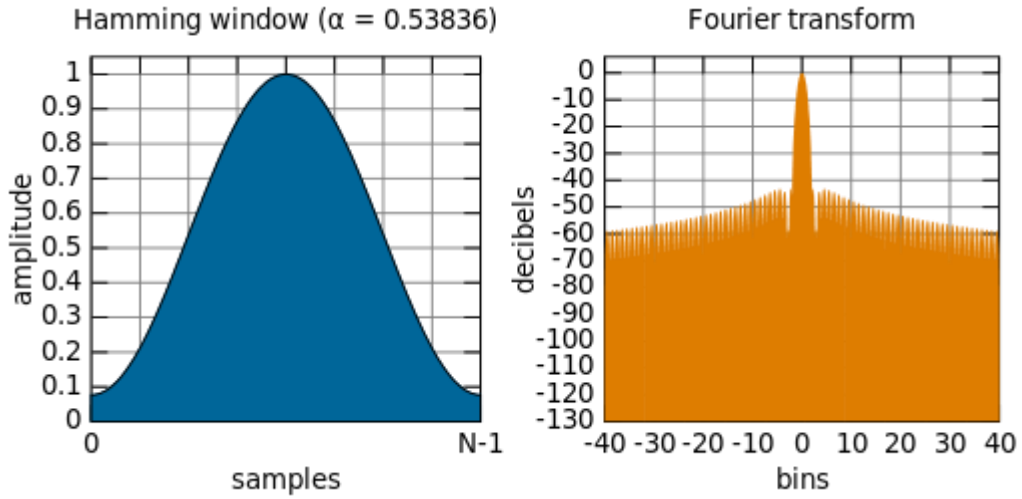


Figura 21. Representación gráfica de la ventana de Hamming [32]

Otras ventanas como la de Von Hann o la de BlackMan han sido descartadas porque se le da demasiada importancia al lóbulo central. La ventana rectangular también ha sido descartada debido a que su función de ventana es igual a 1, es decir, le da la misma importancia a todos los valores de la ventana.

4.2.6 Diseño de la FFT

En cada ventana generada en el apartado anterior se realiza una Transformada de Fourier para convertir la señal del dominio del tiempo al dominio de la frecuencia. Se utiliza la FFT ya que es un eficiente algoritmo que permite calcular la Transformada Discreta de Fourier. De ella solo se necesitan los valores reales y dadas sus características solo será necesario calcular la primera mitad de los, ya que su salida es simétrica, de esta forma se evitará el uso ineficiente de los recursos.

Este módulo del diseño deberá tener como salida un array de dos dimensiones con todas las frecuencias con sus correspondientes potencias.

4.2.7 Cálculo de Potencia Máxima

Este módulo es el responsable de calcular las **tres potencias** más altas de cada ventana con su correspondiente valor de frecuencia. Estos tres pares de valores de cada ventana son almacenados para el siguiente y último modulo.

4.2.8 Diseño de la Comparación e Identificación de sonidos

En este último apartado se toman todos los valores calculados en el apartado anterior y se comparan con los valores característicos de los 5 sonidos estudiados.

Para determinar si el sonido captado coincide o no con el estudiado, se comparan las frecuencias y las potencias características de los sonidos. Estas comparaciones tendrán un error aceptado entre el sonido captado y el estudiado, es decir, las potencias y las frecuencias no tienen que ser exactamente iguales para que superen la comparación, habrá un rango de valores en el que la comparación será válida.

Se le da una mayor importancia a las frecuencias (especialmente a la más alta) sobre las potencias, para ello se usará un sistema de ponderación en el cual cada comparación tendrá una puntuación en caso de ser pasada con éxito. Para que un sonido sea válido, después de realizar todas las comparaciones tendrá que llegar a una puntuación mínima. En el caso de que sea así y sea válido, aparecerá una notificación luminosa en forma de icono en la pantalla del dispositivo

5 Desarrollo

En el actual apartado se relatará el proceso de desarrollo que se ha realizado tras la definición de los requisitos y la descripción de diseño que define la aplicación. En las próximas páginas se describirá, de forma detallada, la totalidad del proceso de desarrollo: la metodología seguida con los lógicos problemas de un desarrollo de este tipo y las correspondientes soluciones que se han ido encontrando. El apartado ha sido dividido por módulos, para facilitar la lectura, con una estructura similar a la que se ha seguido en el apartado 4.2 de Diseño de la Solución.

5.1 Introducción

El desarrollo de este trabajo ha seguido las directrices marcadas en el apartado 4 de Diseño de la Aplicación.

Teniendo en cuenta los conocimientos de los que se partía, la aplicación resulta inicialmente complicada de desarrollar. Pero siguiendo la premisa de dividir el objetivo final en objetivos más pequeños y asequibles, se consigue hacer el trabajo más sencillo. Estos objetivos más pequeños van en creciente dificultad y de manera incremental, es decir, cada nuevo objetivo empieza donde acaba el anterior.

Este desarrollo incremental ha seguido el orden de los módulos que van a ser relatados en las siguientes páginas. Además de perseguirse un desarrollo incremental, también se ha buscado que en cada parte o módulo del desarrollo se pueda probar el incremento de una manera unitaria para así ganar robustez en la aplicación y localizar los fallos de una manera más rápida y eficaz.

Asimismo se ha desarrollado primero una "versión de pruebas" que realiza una única identificación cuando se pulsa el botón de inicio. Esta primera versión se ha implementado con el objetivo de simplificar y agilizar el proceso de desarrollo a la hora de realizar las pruebas de la aplicación. Una vez que se ha realizado todo el desarrollo y se ha logrado una buena calidad en la identificación de sonidos, para la versión final, se ha agregado la funcionalidad de realizar grabaciones secuenciales hasta que se identifique un sonido como se describía en el diseño.

5.2 Grabación de Sonidos

El objetivo de este primer módulo es realizar una grabación con el dispositivo móvil en un formato en el que después se pueda hacer un análisis de audio. Para ello se ha realizado un proceso de documentación acerca de formatos y métodos para grabar audio en *Android*.

Para realizar programas que graben en *Android* el primer paso es dar permisos a la aplicación para utilizar el micrófono y usar la memoria, en caso de que la grabación se quiera almacenar en la memoria interna o tarjeta SD.

Esto se realiza modificando el fichero XML *AndroidManifest* del proyecto *Android*. Este fichero se encarga de recoger todos los permisos, actividades y servicios que controla la aplicación así como la mínima versión de *Android* que utiliza.

En *Android* existen dos clases de su API realmente útiles para realizar grabaciones de audio: *AudioCapture* y *AudioRecord*. Aunque ambas comparten algunos métodos, *AudioCapture* posee métodos de más alto nivel y es utilizada especialmente para realizar grabaciones de formatos con compresión. *AudioRecord*, en cambio, permite operar a más bajo nivel con la información recogida por el micrófono ya que permite operar hasta nivel de bytes y se puede utilizar para grabar con compresión o sin ella. Como se ha relatado en el apartado 4.2.3 del Diseño de la Aplicación, para que sea posible un procesado posterior, es altamente recomendable que la información del audio esté en un formato *raw* o crudo, sin compresión y sin pérdidas. Este requerimiento se podría cumplir con ambas clases, pero dado que se va a tratar esa información posteriormente, resulta más interesante optar por *AudioRecord* que da la posibilidad de manipular la información recogida de la grabación más fácilmente.

Con el objetivo de facilitar las pruebas posteriores, como se ha dicho anteriormente, se ha optado por realizar una primera versión de la aplicación que difiera ligeramente de la versión final, una "versión de pruebas". Todo el proceso relatado en el apartado 4.2.2 Diseño General de la Aplicación, se realiza una única vez cuando se pulse un botón de inicio, es decir, a diferencia de la versión final en la que una vez pulsado el botón de inicio se lanza un Servicio y se ejecuta cíclicamente hasta detectar un sonido familiar, esta primera versión se ejecuta una única vez.

Pero la inclusión de un botón que solo permita una captura no es el único cambio que se ha realizado para simplificar las pruebas de identificación. En la "versión de pruebas" de la aplicación se ha decidido escribir la grabación, realizada por el teléfono, en memoria en formato sin compresión WAV, a diferencia de la versión final en la que se almacena en buffer para optimizar recursos.

La función de almacenar la grabación en WAV es la de capturar los sonidos, poder almacenarlos y reproducirlos tanto en el dispositivo en el que se realizan las pruebas como en un ordenador ya que ambos son capaces de reproducir dicha información sin necesidad de instalar ningún programa o aplicación. Al tener almacenado el audio será más sencillo tratar esas grabaciones para los siguientes módulos, especialmente a la hora de estudiar las frecuencias de los sonidos en el proceso de identificación.

En la siguiente captura se puede observar que existe un botón con un icono de un micrófono, el cuál, al pulsarse se realiza una grabación de 2,5 segundos.

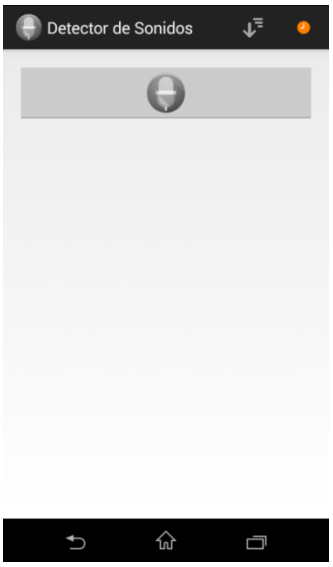


Figura 22. Actividad Principal de la aplicación de prueba

Estos dos cambios entre la versión con la que se realizan las pruebas y la versión final, son muy sencillos a la hora de realizar la implementación. Por este motivo en las siguientes líneas se describirá la "versión de pruebas" y al final del módulo se expondrán los cambios realizados una vez terminados todos los módulos de la aplicación para llegar a la versión final.

Una vez pulsado el botón la aplicación, lo primero que realiza es preparar el contexto para realizar la grabación. En esta preparación se calcula en memoria un tamaño mínimo de buffer en el que se almacenará en forma de bytes la grabación y se instancia la clase *AudioRecord* pasándole por parámetros los valores característicos de la grabación. Algunos de estos valores característicos han sido detallados en el apartado 4.2.3 del Diseño. A continuación se puede ver la instanciación de la clase:

```
recorder = new AudioRecord(
  AudioSource.MIC,
  22050,
  AudioFormat.CHANNEL_IN_MONO,
  AudioFormat.ENCODING_PCM_16BIT,
  minBufferSize
  );
```

Figura 23. Preparación de la grabación con los valores determinados en el diseño

El primer parámetro define la fuente de la que se obtiene el sonido; se elige el micrófono estándar *MIC*, existen varias configuraciones para el micrófono entre ellas destaca *CAMCODER* que orienta la grabación a la dirección que apunte la cámara del teléfono. Esta opción es descartada porque se espera que cuando el teléfono realice la captación no se esté utilizando, estará en un bolsillo, encima de una mesa, etc. Como no existen alternativas que vayan a mejorar la calidad de grabación se decide utilizar el micrófono estándar *MIC*.

El segundo parámetro *22050* es la frecuencia de muestreo definida en el apartado 4.2.3 del Diseño. Los siguientes dos parámetros, también definidos en el diseño, corresponden al canal utilizado para grabar (*MONO*) y al número de bits por muestra y tipo de codificación utilizada (*ENCODING_PCM_16BIT*).

El último parámetro corresponde con el tamaño mínimo de buffer que se ha calculado justo antes de invocar esta instanciación.

Una vez prepara la grabación, se lanza la misma con un método *start* que lo único que hace es lanzar un *thread* de *Android* en el que se almacena un buffer de bytes con la información de grabación.

Los *threads* en *Android*, al igual que en *Java*, son bifurcaciones del programa que se utilizan para realizar tareas en segundo plano con el objetivo de no saturar el hilo principal. En *Android* son muy utilizados para realizar actividades que son pesadas durante un corto periodo de tiempo. Son muy utilizados porque la actividad del hilo principal no puede estar saturada más de 5 segundos, si no el SO cierra la aplicación por seguridad para sistema.

Para terminar la grabación se utiliza otro método de *stop*. En él se interrumpe el hilo creado por el método *start* y la escritura del buffer, para posteriormente almacenar el buffer en un array de bytes.

Para la versión final se habría terminado este módulo al salvar el buffer en un array de bytes.

En cambio, para la versión de pruebas, es necesario generar un fichero WAV. Un fichero WAV, no es más que un fichero con la información *raw* precedida de una cabecera que indica las características del audio almacenado: número de canales, frecuencia de muestreo, bits por muestra, etc.

La información *raw* o cruda corresponde con la información almacenada en el array de bytes, por lo tanto, para generar el fichero WAV bastará con escribir la información en bytes de la cabecera y a continuación agregar el array de bytes con el audio capturado.

Una vez que ya se tiene el fichero WAV, se almacena en una ruta determinada del teléfono donde se guardan varias versiones de los 5 sonidos grabados, de esta manera es más cómodo realizar pruebas, porque no será necesario grabar los sonidos cada vez que se realice un procesado, bastará con abrir uno de los sonidos almacenados en la memoria.

El fichero WAV que se quiera utilizar será necesario abrirlo y eliminar la cabecera. La cabecera no posee información útil para el procesado, la información interesante se encuentra en el apartado *data* de la siguiente figura. Para que la información esté lista para el procesado a diferencia de la versión final, se tendrá que eliminar la cabecera del array de bytes. Esto será tan sencillo como eliminar los 45 primeros valores del array de bytes (desde la posición 0 hasta la 44 del array) que, como se puede observar en la siguiente figura, son los valores que corresponden con la cabecera:

Figura 24. Estructura de un fichero WAV [33].

Una vez eliminada esta información en el array permanecerá únicamente la información de la grabación y ya se podrá comenzar con la eliminación de silencios.

5.3 Eliminación de Silencios

Este módulo parte de la información guardada en el array de bytes que contiene la información del audio *raw o* cruda capturada por el micrófono.

Como se describe en el apartado 4.2.4 del diseño y debido a que la aplicación está corriendo continuamente se requiere de un componente que sea capaz de eliminar información innecesaria de la grabación para agilizar los módulos siguientes, evitar posibles fallos por valores que no son representativos y, al fin y al cabo, hacer la aplicación más robusta y eficiente.

Para realizar este proceso de eliminación de silencios, se buscaron en la red bibliotecas que pudieran ser incorporadas al proyecto. Esta opción era la más sencilla, ya que únicamente es necesario descargar un fichero *.jar*, copiarlo al directorio de librerías referenciadas de *Eclipse*, y cada vez que sea necesario utilizar una clase o método de la librería, se realizan las importaciones automáticamente.

Desafortunadamente no se encontraron librerías de *Android* que realicen una eliminación de silencios como tal, pero si alguna que haga un pequeño análisis DSP como *musicg* [34], aunque analizando la amplitud de la señal se podría realizar una eliminación de silencios, la utilización de la librería no ayudaría mucho al procesado porque lo que hace la librería se puede hacer por código sin mucha dificultad y, dado que se han encontrado mejores algoritmos que este tan sencillo, se ha descartado la utilización de esta librería.

La siguiente alternativa fue buscar una librería de *Java* compatible con *Android*, esto será posible siempre y cuando la librería Java a importar utilice clases que posee *Android*. Así se encontró *The source-code.biz Java DSP collection* [35], esta librería posee una utilidad denominada *ActivityDetector* (detector de actividad) utilizada para señales, que detecta cuando una señal está por encima de un determinado umbral.

Se decidió implementar para realizar pruebas, pero al incorporarse al proyecto esta librería *Java* no daba los resultados esperados y los métodos que utilizaba no eran sencillos de incorporar a la aplicación, por lo tanto se descartó.

Después de esta última experiencia y dado que los algoritmos encontrados no habían convencido, se decidió buscar información sobre algoritmos para la eliminación de silencios en grabaciones con voz para después implementarlo por código. Se buscó información en eliminación de silencios en voz, porque cuando se intentan aplicar mecanismos de eliminación de silencios, normalmente suelen ser con el objetivo de comenzar un procesado de audio en el que el objetivo final sea reconocer y transcribir la voz de un ser humano. Al ser está la aplicación más extendida en eliminación de silencios, la mayoría de los algoritmos están orientados a ello, pero en la aplicación del presente TFG se podrá utilizar cualquiera de estos algoritmos ya que lo único que se quiere detectar en este módulo es si hay sonido o no, independientemente si es de un timbre o de un ser humano hablando. El objetivo de este módulo es eliminar ruidos e información inútil para el posterior procesado.

Realizando búsquedas en esta dirección, se encontró el *paper* con el título *A New Silence Removal and Endpoint Detection Algorithm for Speech and Speaker Recognition Applications* [36]. En este documento se relata un potente algoritmo para la eliminación de silencios y la detección del final del sonido.

Este algoritmo se presenta en el documento, como un método muy útil para eliminar silencios y ruido de fondo en aplicaciones donde se requiere una eficiente extracción de características.

Este algoritmo es más potente que otros más utilizados como los denominados *Short Time Energy* o *Zeros Crossing Rate*. *Short Time Energy* usa la cantidad de energía para determinar si es sonido o silencio, pero no es específico con el valor que tiene que tener una muestra de la señal para ser considerada silencio o sonido. Por el contrario *Zeros Crossing Rate* determina si es sonido o silencio utilizando un umbral, pero no lo compara con el resto de la señal.

Una vez que se ha estudiado este algoritmo, dado que resuelve el problema que demanda la aplicación, se decide implementar. Se encuentra el blog [37] donde aparece código *Java* de dicho algoritmo, dado que no se apoya en ninguna librería externa ni propia de *Java* es totalmente válido para ser incorporado al proyecto *Android*.

El algoritmo se divide en dos partes. La primera parte etiqueta las muestras como silencio o sonido. Posteriormente en la segunda parte se divide la grabación en ventanas no solapadas de 10 ms y determina que ventanas tienen sonido, eliminando de la grabación aquellas ventanas que corresponden a silencios. A continuación, se detallan los pasos que se realizan en este algoritmo:

PARTE 1

1. Se calcula la media (μ) y la desviación típica (σ) de los primeros 200 ms de la grabación, los cuales se consideran silencio. Como la grabación tiene una frecuencia de muestreo de 22 KHz, el número de muestras para realizar estos cálculos será:

$$N = 200\,ms \cdot 22050\,Hz = 4410\,muestras$$

$$\mu = \frac{1}{4410}\sum_{i=1}^{4410} x(i)$$

$$\sigma = \sqrt{\frac{1}{4410}\sum_{i=1}^{4410}(x(i)-\mu)^2}$$

2. Se recorren todas las muestras de la grabación y se comparan en la siguiente inecuación:

$$\frac{x-\mu}{\sigma} > Umbral$$

Si la muestra es mayor que el umbral, esa muestra se considera sonora y se la marca con un 1. Si la muestra, en cambio, es menor que el umbral se considera silencio y se le asigna un 0. El umbral viene determinado por una distribución normal.

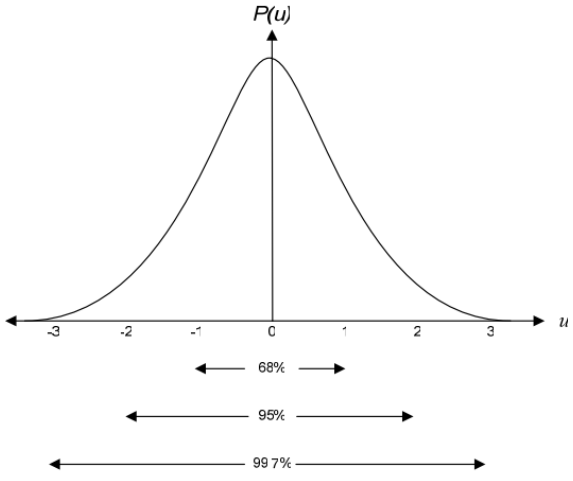


Figura 25. Distribución normal

Un umbral de 3, rechazará el 99,7% de muestras. Con 2, será el 95% y con 1, el 68%. En el caso de querer eliminar silencios que no corresponden con el habla, será necesario rechazar un número muy alto de muestras, pero como se trata de eliminar ruidos y silencios en los que no haya sonido, se utilizará un umbra más bajo. Partiendo de que el umbral debía ser menor de 1, se ha calculado experimentalmente. Tras realizar varias pruebas se ha tomado como valor umbral 0.4.

PARTE 2

Una vez que ya se tiene clasificadas todas las muestras como sonidos o silencio, se realiza la segunda parte de la eliminación de silencios:

1. Se divide toda la grabación en ventanas sin solapar de 10 ms.
2. Si la ventana está compuesta mayoritariamente por 1 se convierte toda la ventana a 1. En cambio, si la ventana posee mayoría de 0 se convierte toda la ventana a 0. De esta forma las ventanas se dividirán en ventanas de 0 (de silencios) y ventanas de 1 (de sonido).
3. Se recolecta y se ordena la información de las ventanas con sonidos (1), correspondiendo con la salida de la eliminación de silencios.

Todo este algoritmo queda descrito gráficamente en el diagrama de actividad de la siguiente página.

Para la adaptación del código a la aplicación, ha sido tan sencillo como incorporar la clase *Java* de [37] en el proyecto. El código de la clase está compuesto con un constructor y un método. En el constructor se le pasan los parámetros necesarios para inicializar el algoritmo:

- La señal de origen: En este caso es la grabación realizada en el módulo anterior, aunque será necesaria convertirla previamente de array de bytes a array de floats. Esto es debido a que el código tiene como parámetro de entrada un array de float. La conversión se hará con un sencillo método que recorra el array transformando cada valor de bytes a un valor float.
- La frecuencia de muestreo: 22 KHz.

A partir de esos dos parámetros se almacenan como variables locales los valores de la señal y la frecuencia de muestreo, y se calculan el número de muestras que hay en los primeros 200 ms y el número de muestras por *frame* (muestras por milisegundo) para posteriores cálculos dentro del método de la clase.

```
public EndPointDetection(float[] originalSignal, int samplingRate) {
    this.originalSignal = originalSignal;
    this.samplingRate = samplingRate;
    samplePerFrame = this.samplingRate / 1000;
    firstSamples = samplePerFrame * 200;
}
```

Figura 26. Constructor de la Eliminación de Silencios

El método de la clase, lo único que hace es lanzar el algoritmo con los pasos detallados en el siguiente diagrama del código. Aquí ha sido necesario modificar el umbral por el cual se considera una muestra en silencio o con información. También ha sido necesario cambiar la longitud de la ventana a 10 ms como se indicaba en [36].

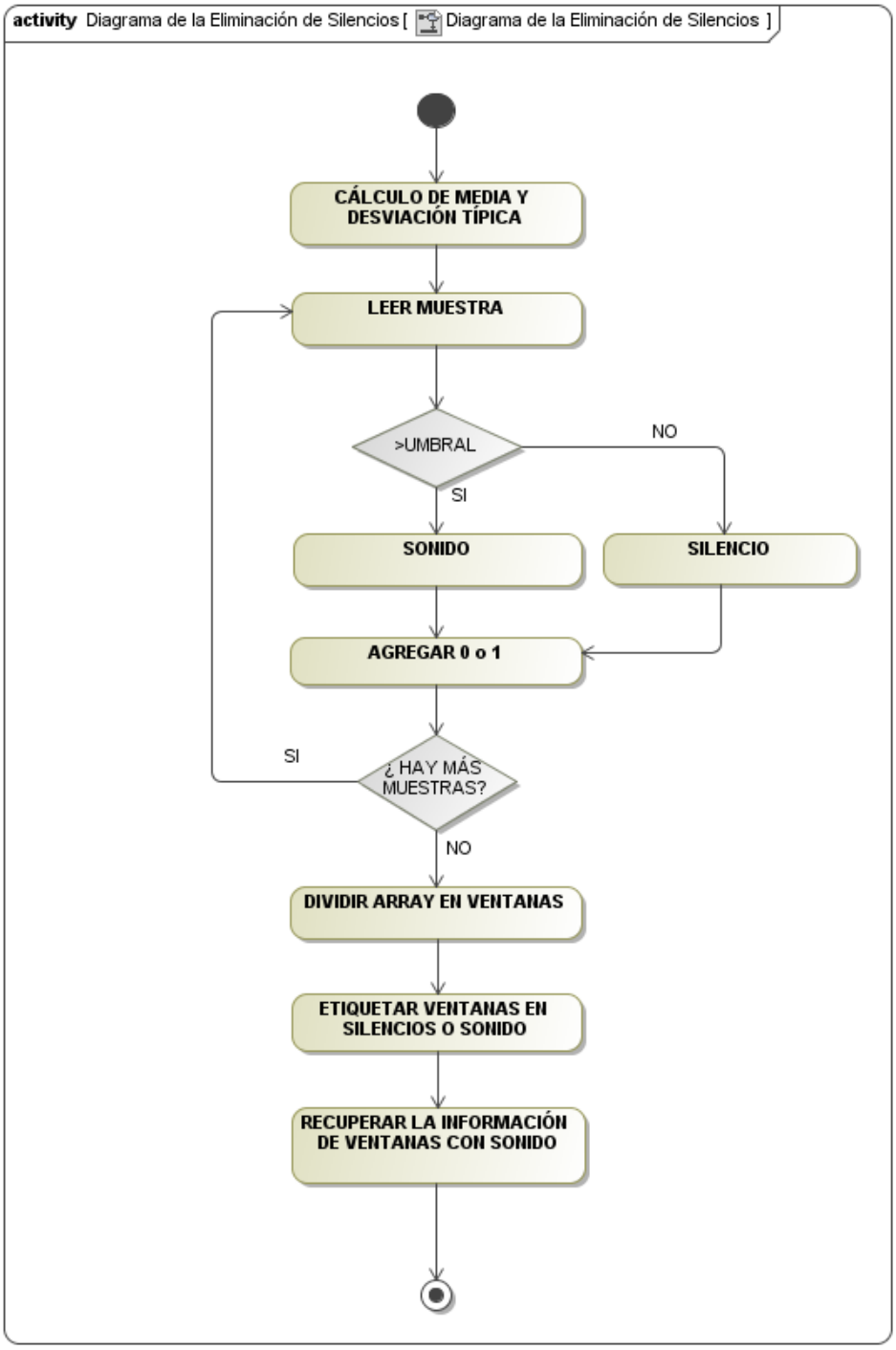


Figura 27. Diagrama del algoritmo para eliminar silencios

5.4 Enventanado

Una vez que se ha realizado la eliminación de los silencios de la grabación se realiza el enventanado que ya ha sido detallado su diseño en el apartado 4.2.5. Al igual que en anteriores apartados, se empezó buscando una librería que ayudara a realizar la operación de enventanado. Se encontraron varias alternativas en diversos lenguajes de programación, ninguno de ellos en java, entre las que destaca *Aquila Open Source DSP library C++* [38]. Esta librería no solo ayuda a realizar el enventanado, posee entre sus características la posibilidad de aplicar enventanado de *Hamming*, sino que también da la posibilidad de realizar una FFT, filtrado de frecuencias incluso extracción de características. Todas estas propiedades la hacían realmente interesante para lo requerido por la aplicación. El problema es que estaba escrita en código C++ lo cual complicaba mucho portar la biblioteca a *Android*. Para ello sería necesario la incorporación de algún tipo de *framework* que convirtiera el código y la librería C++ a *Java* y, las librerías de *Java* fueran compatibles con *Android*.

El enventanado en sí no entrañaba mucha dificultad, pero la idea era buscar una librería que consiguiera realizar varias acciones de la aplicación, en especial para el enventanado, la FFT y la extracción de características. Como las pocas bibliotecas que se encontraron que realizaran las tres acciones no convencieron o resultaban complicadas de implementar, se decidió implementar este módulo por código y para las siguientes fases reanudar la búsqueda de alguna biblioteca que pudiera ayudar especialmente aquellas tareas más difíciles o largas de codificar y que su implementación suponga una considerable mejora de rendimiento.

A este apartado después de la eliminación de silencios se espera, como se expuso en el diseño, que lleguen alrededor de 1,5 segundos de grabación. Como la frecuencia de muestreo es de 22 KHz:

$$N_{Muestras\ Total} = 1{,}5\ seg \cdot 22\,050\ Hz = 33075$$

Y como cada ventana tiene una duración de 0,4 segundos:

$$N_{Muestras\ Ventana} = 0{,}4\ seg \cdot 22\,050\ Hz = 8820$$

Las muestras de una grabación con información estarán en torno a 33000. Para evitar que se realice procesado de grabaciones con pocas muestras se ha decidido que cuando una grabación posea menos de 13230 muestras (dos ventanas) se aborte el procesado y se vuelva al proceso de grabación.

Posteriormente se cuenta el número de ventanas que posee la grabación, para ello se utiliza la ecuación ya descrita en el diseño:

$$N_{VENTANAS} = \frac{DURACIÓN\ DE\ LA\ GRABACIÓN}{DURACIÓN\ MEDIA\ VENTANA} - 1$$

Como cada ventana tiene 8820 muestras:

$$N_{VENTANAS} = \frac{MUESTRAS\ DE\ LA\ GRABACIÓN}{4410} - 1$$

Una vez calculado el número de ventanas, que variará de una grabación a otra dependiendo de la cantidad de silencios que se hayan eliminado en el apartado anterior, para terminar este módulo, se aplicará la función de enventanado a cada muestra de cada ventana.

Como las ventanas están solapadas, para recorrer todas las muestras de la grabación, se ha seguido la estrategia de, a partir de la segunda ventana restar 4410 muestras a donde termina la ventana anterior, para desde ahí aplicar el siguiente enventanado. Este es el algoritmo más sencillo que se ha conseguido para recorrer las muestras a lo largo de las ventanas.

En cada muestra se aplica la función de enventanado que se había mostrado en diseño:

$$w(n) = \alpha - \beta \cos\left(\frac{2\pi n}{N-1}\right)$$

Dónde:

$$w(n) = Valor\ de\ salida\ tras\ aplicar\ la\ función\ de\ enventanado$$

$$\alpha = 0{,}54$$
$$\beta = 0{,}46$$

$$n = Número\ de\ muestra\ en\ la\ ventana$$

$$N = Número\ total\ de\ muestras\ en\ la\ ventana = 8820$$

La función de enventanado de cada ventana se ha implementado con un sencillo bucle *for*:

```
for(i=0;i<=8819;i++)
{
  removalwindow[i]=(0.54 - ((0.46)*(Math.cos((2*(Math.PI)*i)/(8820-1)))))*SRdoub[j];
  j++;
}
```

Figura 28. Código de la función de enventanado

A continuación se puede observar gráficamente lo que se ha descrito en las líneas anteriores:

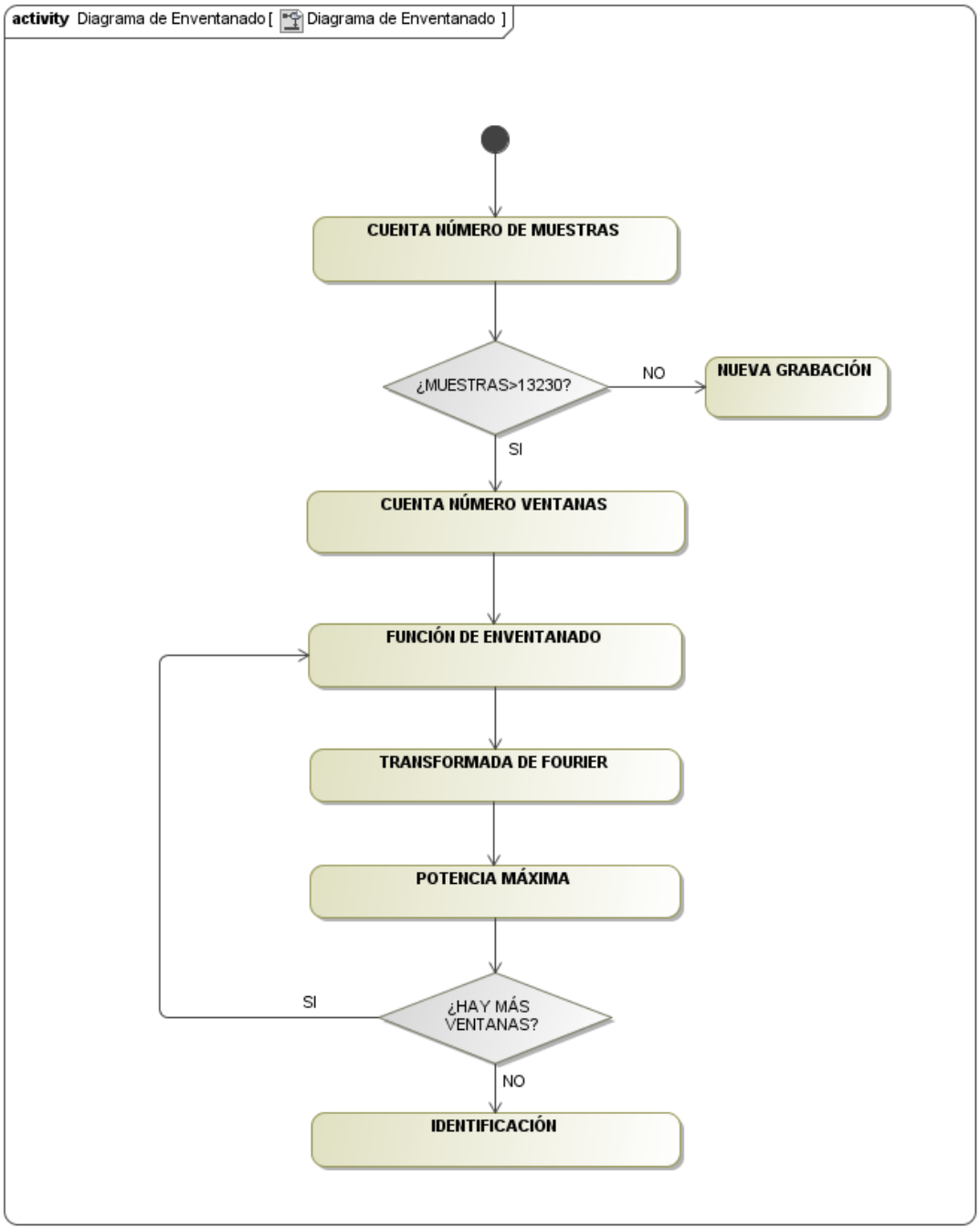


Figura 29. Diagrama detallado del Enventanado

5.5 FFT

Dentro de cada ventana, una vez se ha aplicado la función de enventanado, será necesario pasar la señal del dominio del tiempo al dominio de la frecuencia. Para realizar esta conversión, el mejor algoritmo existente es la Transformada de Fourier. El proceso de realizar una transformada de Fourier dentro de ventanas es denominado también *STFT (Short Term Fourier Transform)* y existen algoritmos que lo implementan, pero por simplificación se ha optado por hacer el enventanado, y dentro de la ventana aplicar la *FFT*.

Para implementar la Transformada Rápida de Fourier, como en módulos anteriores, se buscaron librerías que hicieran esta transformación. En este caso, como es un algoritmo muy conocido y utilizado en tratamiento de señales, se encontraron más y mejores alternativas que en módulos anteriores. Esta *FFT* es mucho más utilizada que algoritmos como los de eliminación de silencio o de enventanado, no solo en tratamiento de señales, sino que también se utiliza en procesados de imagen. Que esta transformación sea tan utilizada favorece la existencia de un gran número de alternativas para la utilización en diversas plataformas.

Entre todas las alternativas disponibles, para el presente trabajo resultaron interesantes las librerías *FFTW* [39] y *JTransforms* [27]. Ambas muy completas, *FFTW* escrita en *C*, mientras *JTransforms* en *Java*. También se encontró código *Java* de *FFT* pero se descartó portarlo por ser menos eficiente y más complejo de implementar.

Se decidió utilizar *JTransforms*, porque al estar escrita en *Java* es muy sencilla de implementar además posee unos métodos muy sencillos y buenas puntuaciones en los test o benchmarks.

Entre las características de *JTransforms* la más interesante para el trabajo que nos trata, es la posibilidad de realizar una transformación con entradas solo de valores reales, lo cual mejora el rendimiento en un 40% respecto a la inclusión de valores imaginarios. Esto resulta realmente interesante ya que tras la eliminación de silencios solo se tienen valores reales y en la mayoría de librería y códigos, era obligatorio introducir valores imaginarios lo cual suponía una utilización poco provechosa de los recursos.

5.5.1 Implementación de *JTransforms*

Una vez elegida la librería *JTransforms*, se implementa. Como se acaba de explicar en este apartado, la librería resulta muy sencilla de integrar en el proyecto *Android*. Para ello se realizan las siguientes tareas:

1. Se le indica a la librería que se va a introducir un array de 1 dimensión cuya longitud corresponde a la longitud de la ventana, es decir, la longitud del array son 8820 muestras.
2. Se inicializa un array de doubles con una longitud del doble de la longitud de la ventana, es decir, 17640 valores o muestras. Esto es debido a que la FFT genera un valor real y otro imaginario por cada valor de entrada.
3. Se realiza la transformada con el método *realForward*, que permite la mejora de rendimiento mencionada anteriormente, y se copia en el array que ha sido inicializado en el punto 2.

5.5.2 Obtención de Resultados y Supresión de Frecuencias

Ya realizada la transformada, es necesario adecuar los valores que genera para poder trabajar con ellos.

El primer paso es, a partir del array de doubles generado a la salida de la FFT, generar los valores de potencia y frecuencia a lo largo del espectro de frecuencias. Para ello se han desarrollado dos métodos: uno que calcula los valores de las frecuencias y otro que calcula las potencias.

Los valores que se obtienen de la *FFT* se dividen en reales y en imaginarios. El primer valor que da la salida es el valor de la potencia en continua, es decir, en 0 Hz. El segundo valor corresponde al último valor real y el resto de entradas del array corresponden a valores reales e imaginarios intercalados entre sí.

Para el cálculo de la potencia se utiliza la ecuación:

$$Potencia(i) = \sqrt{Re^2 + Im^2}$$

Convirtiendo a unidades logarítmicas:

$$dB = 10 \cdot \log_{10} Potencia$$

El método calcula la potencia recorriendo el array de salida de la FFT, obteniendo los valores por pares (1 real - 1 imaginario) para realizar el cálculo de valores siguiendo la ecuación anterior. Como la transformada de Fourier genera un espectro de frecuencia simétrico, se recorre únicamente la mitad del array de salida para evitar tener información repetida y mejorar el rendimiento de la aplicación.

Los dos métodos, de cálculo de potencia y frecuencia, tendrán como salida un array double con la misma longitud. Esto es debido a los valores están ordenados por orden y los valores de frecuencia y potencia están relacionados entre sí por el índice del array, es decir, los pares de elementos potencia-frecuencia están relacionados por el índice del array que ocupan.

Por lo tanto la longitud del array que generan los métodos será:

$$Longitud\ Array = \frac{\frac{Longitud\ Salida\ FFT}{2} - 2}{2}$$

La longitud de salida de la FFT se divide entre 2 porque a cada valor de potencia le corresponden 2 valores de la FFT. Se resta por 2 porque los dos primeros valores no se toman por ser la continua y el último valor. Y todo se vuelve a dividir entre 2 por la propiedad de simetría que tiene la FFT.

Para el cálculo de la frecuencia se utiliza la ecuación:

$$frecuencia(i) = \frac{i \cdot FRECUENCIA\ DE\ MUESTREO\ (Hz)}{MUESTRAS\ DE\ LA\ VENTANA}$$

Dónde:

$$FRECUENCIA\ DE\ MUESTREO = 22050\ Hz$$

$$MUESTRAS\ DE\ LA\ VENTANA = 8820$$

Una vez obtenidas las frecuencias con su correspondiente potencia se decide eliminar aquellos rangos de frecuencias en los que no se van a identificar sonidos, para evitar que en el próximo módulo se calculen frecuencias máximas que no corresponden con las de los sonidos estudiados. Las frecuencias de estos sonidos se encuentran entre 400 - 7000 Hz en la mayoría de los casos, pero como puede haber algún valor que salga de este rango, y para trabajar del lado de la seguridad, se ha decidido ampliar el rango a 200 – 8000 Hz y así evitar que la detección falle en este componente.

5.6 Cálculo de Potencia Máxima

Antes de terminar cada ventana será necesario calcular las potencias máximas de cada ventana y almacenarlas. Para ello se utilizará una función que calcule estas frecuencias máximas. Esta función tiene como entradas los array de potencias y frecuencias obtenida del apartado anterior y, como tercer parámetro, se le indicará si se quiere calcular la frecuencia más alta, la segunda más alta o la tercera más alta. La función recorrerá todas las muestras de los arrays y devolverá una dupla de valores con la potencia máxima y su frecuencia. Al calcular la segunda y tercera potencia se añadirá una restricción extra: será necesario que las tres frecuencias máximas estén separadas por, al menos 200 Hz. Esta restricción se ha incluido debido a que en determinados sonidos las tres frecuencias más altas estaban muy juntas en el espectro de la frecuencia para dar información útil en la posterior identificación.

La función se llamará tres veces para calcular las tres frecuencias más altas y en cada ventana se irán almacenando estas nuevas tres frecuencias (con su correspondiente potencia) al final de otro array con las tres frecuencias máximas de todas las ventanas.

5.7 Comparación e Identificación de Sonidos

Cuando ya se han recorrido todas las ventanas se habrá acumulado la suficiente información para realizar la comparación e identificación de sonidos. Como se acaba de explicar en el apartado anterior, por cada ventana se almacenarán tres pares de valores de frecuencias máximas con su correspondiente ventana. Para el proceso de identificación se irán recorriendo estos valores almacenados y se comparará con las frecuencias y potencias características de los cinco sonidos que acepta la aplicación.

5.7.1 Estudio de los sonidos

Estas frecuencias y potencias características de los sonidos se han obtenido de manera experimental analizando cada uno de los sonidos. Para ello se ha utilizado la aplicación *FrequenSee* de *Android*. *FrequenSee* muestra en tiempo real un espectro de frecuencias de lo que se está grabando con el micrófono.

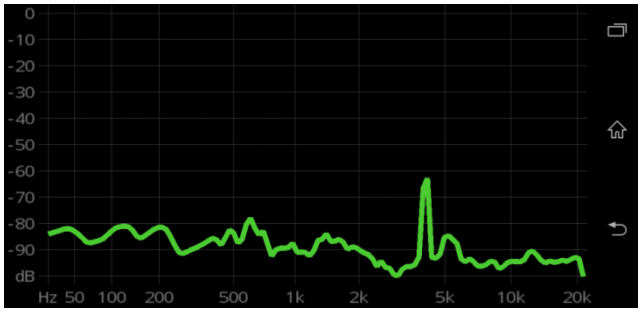


Figura 30. Espectro de frecuencias del sonido de Nevera abierta analizado con FrequenSee

El sonido que hace una nevera cuando se deja la puerta abierta es un molesto pitido, este pitido notifica al usuario que la puerta debe ser cerrada para que no baje la temperatura de la misma. Se ha comenzado estudiando con el sonido de la nevera ya que este pitido es un sonido prácticamente impulsional y por lo tanto, el sonido más simple de analizar. Como se puede observar en la figura anterior, como es un pitido, es prácticamente un sonido impulsivo en 4100 Hz con una potencia en torno a -60 dB. Este par de valores sería la frecuencia y potencia máxima del sonido. La segunda frecuencia máxima, como distan más de 200 Hz de la frecuencia máxima, se situará en torno a los 3800 Hz. Dada la sencillez de este sonido no será necesario realizar un análisis de la tercera frecuencia más importante.

El caso del sonido del fin de microondas es similar al de la nevera. El microondas cuando termina de calentar la comida emite un pitido intermitente. Este pitido también es prácticamente impulsional en 2050 Hz y -40dB. A diferencia de la nevera, tiene una segunda frecuencia máxima característica distante de la principal a 4100 Hz y -70dB.

Aunque el resto de sonidos son más complejos que el de la nevera y el microondas, a la hora de estudiarlos, se ha seguido estrategias similares.

Una vez analizado el fin del microondas se analizan los sonidos del teléfono y el telefonillo los cuales a diferencia de los anteriores tienen varias frecuencias importantes.

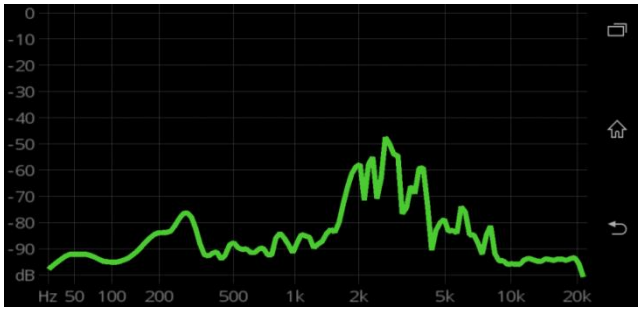


Figura 31. *Espectro de frecuencias del sonido del Teléfono abierta analizado con FrequenSee*

En el sonido del teléfono se puede observar que aparece una frecuencia máxima en 3000 Hz y -50dB. Como segunda y tercera frecuencia máxima aparecen en 2700 Hz y 2000 Hz respectivamente, ambas con una potencia entorno a -55dB.

Para el telefonillo aparecerán 4 frecuencias importantes: La frecuencia máxima 3100 Hz, -40dB. La segunda más importante con 3500 Hz, -45dB. Y la tercera con 4600 Hz. Nótese que como las frecuencias son más importantes en la comparación que las potencias, estás solo serán importantes para la frecuencia máxima y en algún caso excepcional para la segunda frecuencia más alta.

Para finalizar se ha estudiado el sonido más complejo que va a ser tratado en este TFG: el timbre (Ding-Dong). La principal dificultad que tiene este sonido es que posee dos tonos. Como aparece en la siguiente figura, primero suena un tono en 900 Hz aproximadamente (Ding) y luego suena el segundo tono en 780 Hz (Dong). Además aparece una frecuencia secundaria importante que se sitúa entorno a los 4-5 Khz.

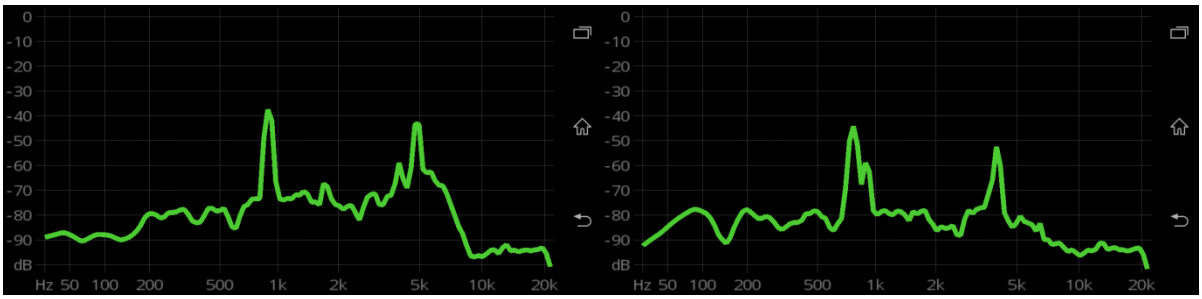


Figura 32. Espectro de frecuencias del Timbre de la Puerta analizado con FrequenSee

5.7.2 Algoritmo de Comparación

El algoritmo de comparación se basa en un sistema de puntuaciones. Se recorrerán los arrays de las frecuencias máximas obtenidas y se irán realizando comparaciones con los valores característicos de los 5 sonidos. Se le asignará un valor a cada comparación, en caso de que la comparación sea satisfactoria, este valor se le sumará a una variable del sonido, si al finalizar el array de frecuencias máximas el valor de la variable es mayor que 10, se considerará dicho sonido como identificado.

La comparación se realiza aceptando un rango de valores cercanos a los estudiados en el apartado anterior. Para ello se aceptará un margen de error del 5% para las frecuencias y un 20% para las potencias. Se empezó con un margen de error mayor para las frecuencias pero tras realizar varias pruebas se lograban valores muy exactos y, al ser más decisivas para la comparación, se decidió ser más restrictivo.

En todos los sonidos existirá una primera condición más restrictiva que sea indispensable para lograr puntuación. En la mayoría de los casos será la frecuencia principal, ya que esta frecuencia es muy característica del sonido.

```
//TELEFONO
if(((Math.abs(3000 -(fventanas1[i][0])))/3000)< 0.05)
{
    if(((Math.abs(50 -(fventanas1[i][1])))/50)< 0.2)
        puntTelefono=puntTelefono +1;
    if((((Math.abs(2000 -(fventanas2[i][0])))/2000)< 0.05)||(((Math.abs(2000 -(fventanas3[i][0])))/2000)< 0.05))
        puntTelefono=puntTelefono +5;
    if((((Math.abs(2700 -(fventanas2[i][0])))/2700)< 0.05)||(((Math.abs(2700 -(fventanas3[i][0])))/2700)< 0.05))
        puntTelefono=puntTelefono +5;

}
```

Figura 33. Algoritmo de comparación para el sonido del Teléfono

En la figura anterior se puede observar el algoritmo de comparación. En este caso, la propiedad más restrictiva que corresponde con el *if* más externo y la frecuencia más alta, 3 KHz. Para que haya puntuación será necesario que el valor obtenido de frecuencia máxima esté cerca de 3 KHz y además se cumpla una de las 3 condiciones restantes:

- La potencia máxima deberá tener un valor de -50dB con un error del 20% como se ha mencionado antes, en caso de ser correcto, se le sumará un punto a la puntuación.
- La segunda o tercera frecuencia máxima sea 2 KHz, en ese caso se sumarán 5 puntos a la puntuación final.
- La segunda o tercera frecuencia máxima sean 2700 Hz, en ese caso se sumarán 5 puntos a la puntuación final.

Como valores de la segunda y tercera frecuencia máxima se aceptan 2 KHz y 2,7 KHz indistintamente porque se ha comprobado de manera experimental que, al tener potencias parecidas se pueda detectar como segunda frecuencia máxima o tercera.

Por lo tanto para que sea reconocido el teléfono se tienen que cumplir, al menos, dos condiciones de 5 puntos y una de un punto. A la potencia se le ha dado solo un punto para que por sí sola no pueda dar la captación positiva.

Los algoritmos de comparación del resto de sonidos, salvo del timbre de la puerta, serán muy similares al detallado anteriormente. El timbre de la puerta tendrá la particularidad de que será necesario reconocer primero la frecuencia máxima del "Ding" (900 Hz) antes de reconocer el "Dong" (780 Hz). Para ello, se ha implementado, mediante la ayuda de una variable booleana, un mecanismo que para que tenga puntuación la comparación al reconocer la frecuencia máxima del "Dong" haya tenido que reconocer antes el "Ding". Esta es la única comparación que se ayuda del espectro del tiempo para identificar un sonido, ya que el array de potencias máximas guarda las frecuencias máximas en el orden de las ventanas.

5.7.3 Notificación

Cuando la aplicación identifique un sonido, se lanzará una notificación con un icono identificativo. Si el teléfono está bloqueado o con la pantalla apagada, se ejecuta un *WakeLock*, que sirve para "despertar" el procesador. Después de ejecutar el *WakeLock*, se desbloquea el teléfono. Esto se logra dando permisos de *WakeLock* y de desactivar bloqueo en el fichero *AndroidManifest.xml*, al igual que se hacía en el módulo de grabación dando permisos a realizar grabaciones. En ese momento la detección se desactiva y la imagen se queda persistente en la pantalla hasta que se pulsa el botón atrás y se vuelve a captar sonidos.

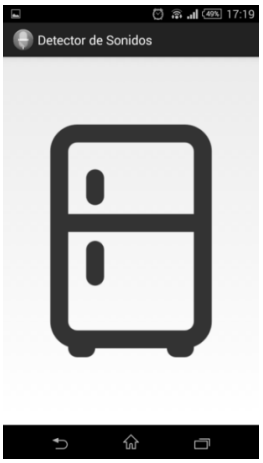


Figura 34. Notificación de puerta de nevera abierta

5.8 Lanzamiento en Modo Inicio y Final Automático

La aplicación posee un modo para activar la identificación de modo automático en un rango de horas elegido por el usuario. Se podrá configurar el inicio y el final o bien uno de los dos. Se podrá configurar una hora de inicio y posteriormente parar la detección manualmente. Asimismo se podrá iniciar la aplicación manualmente pero que el fin sea automático.

Para ello la aplicación dispone de un menú en el que se puede configurar el inicio y el final de manera independiente.

En ambos aparece, marcada por defecto, una casilla de inicio/fin manual, cuando esta se desactiva y se introduce una hora, se inicia el modo automático del inicio/fin, según el menú que se haya modificado.

La gestión del inicio y el final automático se ha realizado con los *Service* de *Android*, que son servicios ligeros que corren en segundo plano.

Para el caso del inicio automático, cuando se introduzca la hora en el menú de inicio automático se realizarán las siguientes acciones:

1. Se comprobará que no se están identificando sonidos, si se estuviera identificando sonidos, no se lanzaría el servicio porque ya se habría iniciado manualmente la captación de los mismos.
2. Si no se están captando sonidos, se lanza el servicio con la hora de inicio de la captación. El servicio corre en segundo plano de manera persistente indistintamente si la aplicación está abierta o no.
3. Cuando llega la hora de inicio de la captación, el fin del servicio es recibido por un *BroadcastReceiver* de *Android*. Los *BroadcastReceiver* son mecanismos para recibir notificaciones dentro de una aplicación. El *BroadCastReceiver* se encarga de iniciar la captura de sonidos y de lanzar el servicio de final automático en caso de que este activado el final automático en el menú de configuración.

Para el caso de fin automático sería muy similar:

1. Cuando se introduzca la hora en el menú de fin automático, se comprobará que se están identificando sonidos, si no se estuviera identificando sonidos, no se lanzaría el servicio.
2. Se lanza el servicio con la hora de fin de la captación. El servicio corre en segundo plano.
3. Cuando llega la hora de fin de la captación. el fin del servicio es recibido por un *BroadcastReceiver*. El *BroadCastReceiver* se encarga de finaliza la captura de sonidos y de lanzar el servicio de inicio automático en caso de que este activado el inicio automático en el menú de configuración.

Para volver a activar el modo manual, bastaría con marcar las dos casillas de inicio/fin manual. Esto desactiva los servicios de inicio/fin automático y la captación de sonidos solo podrá ser activada con el botón.

5.9 Desarrollo de Mejoras de Rendimiento

Una vez finalizados todos los módulos del desarrollo de la aplicación, y esta ya es capaz de reconocer sonidos, se han implementado medidas para mejorar el rendimiento. Al finalizar el módulo de Comparación e Identificación de Sonidos (5.7) la aplicación requiere de muchos recursos mientras está captando sonido. Especialmente la aplicación utiliza mucha memoria RAM, acercándose en exceso al máximo de memoria que *Android* da a cada aplicación. El tamaño de esta memoria varía en función del dispositivo que se está ejecutando. Si una aplicación supera el límite de la memoria en algún momento, *Android* con su utilidad *Garbage Collector*, se encarga de matar los procesos más pesados.

Se hicieron algunas pruebas y la aplicación pasado el tiempo se paraba por el motivo que acaba de ser descrito. Tras esto, el primer objetivo fue entender cómo funciona el *Garbage Collector* de *Android*. Se descubrió que *Android* hacía un manejo de memoria distinto según la versión mínima de la aplicación desarrollada. A partir de las versión 4.0 (API 15), la gestión de memoria cambia y es más eficiente, a eso hay que añadirle que en esa versión se añade la posibilidad de insertar en el fichero *AndroidManifest.xml* una declaración para aumentar la cantidad de memoria que va a disponer la aplicación. Por este motivo se decidió subir la versión de *Android* a la versión 4.0, contradiciendo lo propuesto en el apartado de diseño (4.2.1). Subiendo la versión se pierden potenciales usuarios del 99,3% hasta el 85,8% pero se mejora sustancialmente el rendimiento. De cualquier forma, el ritmo al que las versiones antiguas están dejando de ser utilizadas es muy alto, y la tendencia es que en el próximo año ese valor se haya acercado considerablemente al 100%.

Adicionalmente, se descubrió que el *Garbage Collector* podía ser llamado en el código, en ese momento se introdujeron llamadas al *Garbage Collector* en determinados partes del procesado donde se consideraba que se podía eliminar, una vez utilizados, instancias y arrays que podían resultar pesados para el proceso que realiza la aplicación.

5.10 Desarrollo del Apartado Gráfico y Estadísticas

El apartado gráfico y estadísticas se han desarrollado una vez que se han terminado los anteriores módulos ya que no son apartados de usabilidad de la aplicación, si no estéticos. Esto no debería ser una práctica común pero dado que se trata de una aplicación con muy poco interfaz gráfico, la mayoría de sus operaciones se realizan en servicios sin que el usuario lo perciba, se ha decidido dejar este apartado para el final.

En el ANEXO B se puede ver el interfaz gráfico final de la aplicación además del resultado del apartado Estadísticas.

En las Estadísticas que representa la aplicación aparecen los siguientes apartados:

- **Registrador o Log**: Muestra en pantalla un registro con la fecha y la hora en la que la aplicación ha registrado uno de los cinco sonidos. Para implementar este registrador, la aplicación genera un fichero de texto plano con la extensión .log la primera vez que se inicia, y cada vez que se detecta un sonido se escribe una traza en el fichero con la fecha, hora y el detalle del sonido detectado.

  ***Ejemplo*:**

  20/08/2014 19:23 SONÓ EL MICROONDAS

  Este registro se agrega al final del fichero log y en el apartado de Estadísticas se lee el fichero entero y se muestra por pantalla.

- **Número de sonidos identificados**: Se presentan por pantalla el número de veces que ha sonado cada uno de los cinco sonidos desde que se ha instalado la aplicación. Para implementar este apartado de estadísticas se ha utilizado la propiedad de Android *SharedPreferences* que son contenedores que almacenan información de la aplicación. Se ha creado un *SharedPreference* de tipo entero para cada uno de los 5 sonidos, y cada vez que la aplicación detecta ese sonido le suma una unidad al *SharedPreference* de dicho sonido. En la pantalla de estadísticas simplemente se imprime el valor de los cinco *SharedPreferences*.

- **Gráfica de número de sonidos identificados**: En esta pantalla se muestra un gráfico con los valores del punto anterior. Para ello utiliza los cinco contenedores *SharedPreferences* y la librería HoloGraphLibrary [40] en su vista *LineGraph*. Se trata de una pequeña librería de código abierto para Android en la que se pueden importar las clases .java de una manera sencilla al proyecto.

- **Acerca de:** Se muestra el nombre del TFG, la fecha y los nombres del autor, la titulación y la universidad.

Todos estos apartados han sido implementados en cuatro pantallas con la tecnología *swype*. Es decir, para navegar entre pantallas basta con deslizar el dedo horizontalmente en un sentido o en otro dependiendo de la pantalla a la que se quiera mover el usuario.

6 Pruebas

En este apartado se presentarán las pruebas realizadas para comprobar el funcionamiento de la aplicación creada. Las pruebas se centrarán en probar o testear la capacidad que tiene la aplicación para la identificación de sonidos en distintos entornos. Asimismo, también se va a mostrar un breve apartado en el que se prueba el rendimiento de la aplicación.

Las pruebas se han realizado en un *smartphone* "*Sony Z1 Compact*" con la versión de *Android* 4.4 (*KitKat*). Dadas las características de la aplicación, no se ha optado por probar la aplicación en el emulador de Eclipse, ya que la aplicación requiere del uso de un micrófono y para ello es necesario configurar previamente el emulador. Además de que generalmente el uso del emulador ralentiza el proceso de desarrollo, respecto a depurar la aplicación en un dispositivo, por este motivo el emulador se usa para casos muy concretos. Utilizando un dispositivo para depurar la aplicación, no es necesario realizar ningún tipo de configuración, bastará con activar el modo desarrollador dentro del menú de ajustes del dispositivo.

6.1 Pruebas de Reconocimiento y Resultados

Durante todo el proceso de desarrollo se han realizado las pertinentes pruebas, que han ido comprobando que la inclusión de cada nuevo módulo generaba los valores adecuados. Para ello, se ha utilizado el modo *debug* que ofrece Eclipse: conectado el teléfono al ordenador, se lanza la aplicación en el dispositivo y en el ordenador se puede observar el estado de las actividades y las variables de la aplicación mientras se está interactuando con ella en el dispositivo.

Una vez llegado al apartado 5.7 del desarrollo, correspondiente con la implementación del algoritmo de comparación, se han comenzado a realizar pruebas con el dispositivo desconectado del modo *debug*. Estas pruebas con la aplicación en desarrollo han sido muy similares a las que se van a detallar a continuación, que corresponden a la versión final de la aplicación. En las pruebas se emitía uno de los 5 sonidos y se comprobaba si la aplicación lo detectaba, no lo detectaba o detectaba otro sonido. Al comienzo, para agilizar estas pruebas con la aplicación aún por terminar, se grabaron los 5 sonidos en el ordenador y se reproducían en los altavoces, de esta manera la realización de estas pruebas se agilizó considerablemente.

Una vez terminado el desarrollo de la aplicación se han realizado las pruebas que van a ser relatadas en las siguientes líneas. Las pruebas de reconocimiento para cada sonido se han divido en dos escenarios: en un entorno sin ruidos y en un entorno adverso. Para cada entorno se han realizado 10 pruebas, es decir, 20 pruebas para cada sonido en total. Lógicamente la detección será mejor en entornos sin ruidos que entornos adversos con ruidos de fondo, por este motivo cada prueba de cada sonido se ha dividido en dos partes.

Los ruidos de fondo que se han introducido en estas simulaciones son los típicos sonidos que hay en una casa: una televisión encendida, una plancha, personas hablando, etc. Estos ruidos de fondo no deberán ser mayores en potencia que el sonido a identificar para que la aplicación pueda detectar el sonido. En principio la aplicación debería captar ruidos en entornos adversos pero con algo menos de precisión.

Para representar la precisión de las pruebas se ha utilizado el denominado *F-Score* (Valor-F en castellano). Este test se emplea para dar un único valor ponderado de la precisión y de la exhaustividad [41]. El *F-Score* es muy común en pruebas de identificación y recuperación de información, por este motivo se ha decidido representar los datos de los test con el valor que ofrece este test. Un test exitoso será cuando su valor este cercano a la unidad.

La precisión se define como la fracción de instancias recuperadas que son relevantes, mientras que la exhaustividad, es la fracción de instancias relevantes que han sido recuperadas.

La fórmula general del test es:

$$F_\beta = (1+\beta^2)\cdot\frac{Precisión\cdot Exhaustividad}{(\beta^2\cdot Precisión)+Exhaustividad}$$

Donde β sirve para ponderar a favor de la precisión o de la exhaustividad, valores mayores que la unidad dan más importancia a la exhaustividad, mientras que valores inferiores a la unidad le dan más importancia a la precisión. En el presente TFG, como se consideran igual de importantes ambos valores, se le ha asignado el valor 1 quedando la siguiente ecuación simplificada:

$$F_1 = 2\cdot\frac{Precisión\cdot Exhaustividad}{Precisión+Exhaustividad}$$

Para el cálculo de la precisión:

$$Precisión = \frac{Positivos}{Positivos+Falsos\ Positivos}$$

Y para la exhaustividad:

$$Exhaustividad = \frac{Positivos}{Positivos + Falsos\ Negativos}$$

En los test que se van a realizar a continuación se hace sonar un determinado sonido y se observa cómo reacciona la aplicación, como se ha dicho anteriormente, los resultados podrán ser: *DETECTADO (D)*, *NO DETECTADO (ND)* O *FALSO POSITIVO (FP).*

Trazando los valores de la precisión y la exhaustividad con los posibles valores que habrá en cada prueba, los valores de las ecuaciones anteriores corresponden con los siguientes resultados:

- Positivos = *DETECTADO (D).* Cuando se detecte el sonido que está sonando.
- Falsos Negativos = *NO DETECTADO (ND).* Cuando suene el sonido pero no se detecte.
- Falsos Positivos = *FALSO POSITIVO (FP).* Cuando suene el sonido y se detecte otro sonido.

Las pruebas que se detallan a continuación quedan resumidas en una tabla para cada sonido. La cual recogerá las 10 pruebas en cada entorno, con sus posibles valores D, ND o FP. En cada entorno se calcula la precisión, exhaustividad y Valor-F con las ecuaciones recogidas anteriormente. Asimismo se calcularán estos valores para los dos entornos, dando un resultado final de precisión, exhaustividad y Valor-F de la aplicación para captar ese determinado sonido en todos los entornos.

6.1.1 Detección del Microondas

Se ha comenzado probando el sonido que hace el microondas cuando termina. Como se trata de un sonido prácticamente impulsional, se repite varias veces y el microondas genera un sonido con una elevada potencia acústica, ha tenido un alto porcentaje de detección.

Su rendimiento en entorno adverso ha disminuido sensiblemente, algo lógico dados los diversos sonidos que puede haber en una cocina que alteren la identificación. No se han detectado falsos positivos, por lo tanto la precisión total ha sido del 100%.

Prueba	Entorno Tranquilo	Entorno Adverso
1	D	D
2	D	D
3	D	D
4	D	D
5	D	D
6	D	ND
7	D	ND
8	ND	D
9	D	D
10	D	ND
Precisión	100%	100%
Exhaustividad	90%	70%
Valor-F	94%	82%
Precisión Total	100%	
Exhaustividad Total	80%	
Valor-F Total	88%	

Tabla 1. Resultados de pruebas de detección del Microondas

6.1.2 Detección de la Nevera

Las pruebas con el sonido generado por la nevera al dejar la puerta abierta han tenido resultados similares a los del microondas. Esto es lógico porque son sonidos con características similares: prácticamente impulsionales y con una elevada potencia acústica. Por este motivo los resultados han sido similares al microondas, aunque su robustez frente a los entornos adversos ha sido mayor. Esto puede ser debido a que sus frecuencias características más importantes se sitúan en frecuencias más alejadas y con más potencia que los ruidos que aparecen en la cocina. No se han detectado falsos positivos, por lo tanto la precisión ha sido del 100%.

Prueba	Entorno Tranquilo	Entorno Adverso
1	ND	D
2	D	D
3	D	D
4	D	D
5	D	D
6	D	ND
7	D	D
8	D	D
9	D	D
10	D	D
Precisión	100%	100%
Exhaustividad	90%	90%
Valor-F	94%	94%
Precisión Total	100%	
Exhaustividad Total	90%	
Valor-F Total	94%	

Tabla 2. Resultados de pruebas de detección de la Nevera

6.1.3 Detección del Telefonillo

El siguiente sonido en ser probado ha sido el telefonillo. El telefonillo al tener tantas frecuencias características, cuatro, ha logrado un porcentaje de detención menor. Entre estas cuatro frecuencias no hay una frecuencia máxima clara, lo cual hace que cada detección sea una de las cuatro frecuencias características la frecuencia máxima. No se han detectado falsos positivos, por lo tanto la precisión ha sido del 100%.

Prueba	Entorno Tranquilo	Entorno Adverso
1	D	ND
2	ND	ND
3	D	D
4	D	D
5	ND	D
6	ND	D
7	D	ND
8	ND	D
9	D	D
10	D	D
Precisión	100%	100%
Exhaustividad	60%	60%
Valor-F	75%	75%
Precisión Total	100%	
Exhaustividad Total	60%	
Valor-F Total	75%	

Tabla 3. Resultados de pruebas de detección del Telefonillo

6.1.4 Detección del Teléfono

Las pruebas de detección del sonido del teléfono de una casa son las que han tenido los resultados más satisfactorios, con un 100% de acierto en las 10 pruebas en un entorno tranquilo y un buen porcentaje en un entorno adverso. Esto es debido a que se trata de un sonido cuyas frecuencias principales son 3 y son muy marcadas, además es un sonido con una potencia acústica alta y se repite durante un periodo suficientemente largo como para ser detectado. La mayoría de las detecciones se han realizado en el primero o segundo tono, lo cual daría tiempo suficiente al usuario a llegar a contestar al teléfono.

Prueba	Entorno Tranquilo	Entorno Adverso
1	D	D
2	D	FP
3	D	FP
4	D	D
5	D	D
6	D	D
7	D	D
8	D	D
9	D	D
10	D	D
Precisión	100%	80%
Exhaustividad	100%	100%
Valor-F	100%	88%
Precisión Total	90%	
Exhaustividad Total	100%	
Valor-F Total	95%	

Tabla 4. Resultados de pruebas de detección del Teléfono

6.1.5 Detección del Timbre de la Puerta

Por último, se han pasado las pruebas de detección al timbre de la puerta. Como ya se detalló en el desarrollo, este era el sonido más complejo puesto tenía dos tonos a lo largo del tiempo. Por la complejidad para captar el sonido con las características de la presente aplicación, este ha sido el sonido que peor tasa de éxito ha tenido. Asimismo se trata de un sonido más largo, y dado que la aplicación capta sonidos durante 2,5 segundos y luego los procesa, si el sonido se empieza a detectar al final de la grabación, este no podrá ser detectado, provocando que la capacidad para captar este tipo de sonidos más largos sea mucho menor.

No se han detectado falsos positivos, por lo tanto la precisión ha sido del 100%, esto ha provocado que aunque la exhaustividad haya sido baja, un 45%, haya un índice de *Valor-F* del 62%.

Prueba	Entorno Tranquilo	Entorno Adverso
1	ND	ND
2	D	D
3	D	ND
4	ND	ND
5	ND	ND
6	D	D
7	D	D
8	ND	ND
9	ND	D
10	D	ND
Precisión	100%	100%
Exhaustividad	50%	40%
Valor-F	66%	57%
Precisión Total	100%	
Exhaustividad Total	45%	
Valor-F Total	62%	

Tabla 5. Resultados de pruebas de detección del Timbre de la Puerta

6.1.6 Falsos positivos

Además de los test relatados en el apartado anterior, se ha dejado la aplicación encendida durante largos periodos de tiempo para ver si daba falsos positivos. Estas limitaciones no pueden ser probadas en los test, pero a continuación se relatan cuáles han sido las más significativas.

Dentro de la casa se han dado ocasiones que ha habido falsos positivos en especial con el telefonillo. Se ha detectado el telefonillo cuando sonaba un secador o algún otro sonido con muchas componentes en el rango de frecuencias donde se encuentra registrado el telefonillo.

En entornos tranquilos como una oficina o una calle por la noche, no ha habido falsos positivos. En entornos con mucho ruido como lugares con música ha existido algún falso positivo como microondas. Con el motivo de disminuir el consumo de batería y estos posibles positivos se introdujo el modo de inicio automático de la aplicación (Desarrollo 5.8), el cual sirve para evitar estos falsos positivos fuera del hogar.

6.2 Pruebas de Rendimiento

El rendimiento de la aplicación ha sido probado con la herramienta *DDMS* que provee Eclipse. Esta herramienta sirve para, con el terminal conectado al ordenador, monitorizar el uso que hace la aplicación de las características del teléfono. En la presente aplicación, las pruebas se han centrado en comprobar el uso que se hace de la memoria RAM, dado que es una de sus grandes limitaciones. La prueba de rendimiento ha consistido en lanzar la captación de sonidos de la aplicación y observar la pestaña de *Heap* de la herramienta *DDMS*.

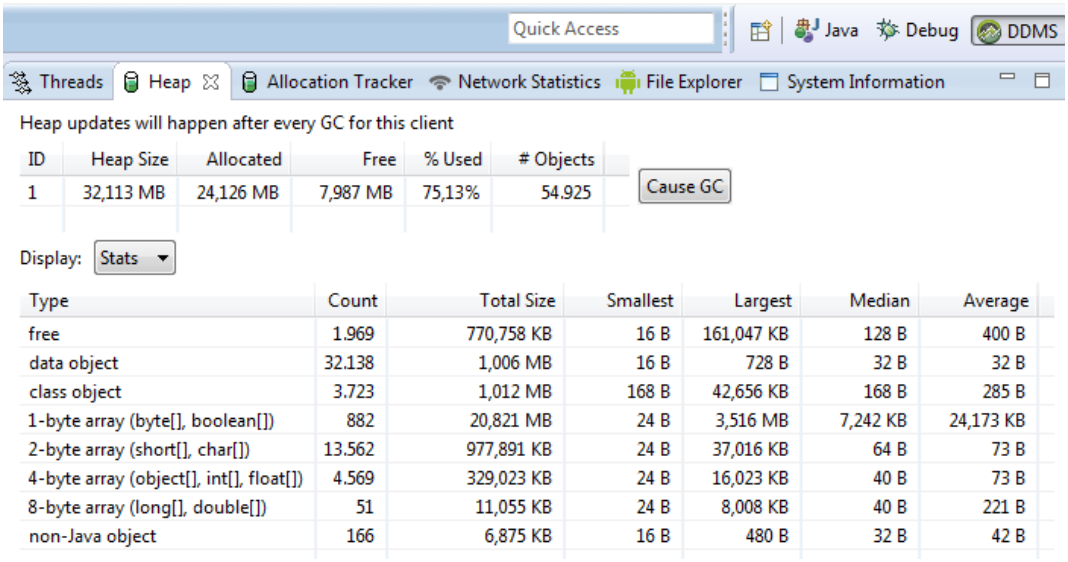


Figura 35. Captura de pantalla de la herramienta DDMS de Eclipse

Para el terminal utilizado en las pruebas, *Android* da un máximo de 32Mb como se puede observar en el valor de *Heap Size* en la anterior figura. El dato más importante que aparece en la captura, es el porcentaje de uso de la memoria: 75%. Este ha conseguido ser bajado del 90% que alcanzaba en las primeras versiones de la aplicación, antes de haber realizado el desarrollo de las mejoras del rendimiento (5.9). Resulta importante esta disminución del porcentaje de uso de memoria para así evitar que se pare la aplicación cuando lleve un largo tiempo ejecutándose.

También destaca de la figura, como la mayoría del uso de la memoria (22MB aproximadamente) es por uso de arrays, que son los que almacenan la información que está grabando la aplicación y a su vez está procesando.

Para finalizar las pruebas de la aplicación, se ha realizado una pequeña prueba en el dispositivo: se ha dejado corriendo la aplicación durante un largo periodo de tiempo en modo de detención y sin utilizar el terminal, este ha sido el resultado:

Figura 36. Captura de pantalla del menú de aplicaciones Android

Tras 6 horas y 43 minutos ejecutándose el proceso de captación, la aplicación solo ha supuesto un 3% en el uso de la batería total del teléfono, es decir, la utilización de la aplicación no ha supuesto un gasto extra muy grande en el uso de la batería. Hay que tener en cuenta que ha sido la primera aplicación que no forma parte del sistema de *Android* en gasto de batería, pero para el proceso tan pesado en gasto de recursos como el de realizar grabaciones y procesados continuamente, se puede considerar que se ha logrado una aplicación eficiente en la utilización de recursos.

7 Conclusiones y Líneas Futuras

7.1 Conclusiones

En el presente TFG se ha demostrado que es posible implementar un detector de sonidos en un *smartphone* con el fin de ayudar a personas con problemas de audición. Para ello ha sido necesario desarrollar una aplicación *Android* que se apoye en conceptos de procesado de audio y patrones de identificación de sonidos.

Como se ha podido observar en el capítulo de pruebas, se ha logrado detectar sonidos en todo tipo de condiciones con un alto nivel de acierto en la mayoría de los casos, ya que el timbre de la puerta ha tenido una baja tasa de acierto. Esto es debido a que se trata de un sonido más complejo y lento de detectar, probablemente se podría haber mejorado la detección de este sonido aumentando el tiempo que se está grabando un sonido (2,5 segundos) pero esto hubiera repercutido en un mayor gasto de memoria en la aplicación y, en este caso, se ha decidido prevaler el rendimiento sobre la capacidad de detección, ya que para el resto de sonidos no era necesario aumentar el tiempo de grabación.

Asimismo se ha logrado desarrollar una aplicación eficiente en el uso de recursos, que es capaz de estar captando sonidos durante horas sin que esto suponga un consumo de batería excesivo, esto resulta realmente interesante dado lo limitadas que son las baterías de los smartphones hoy en día. En este sentido, también se ha aportado la solución de usar un temporizador para activar automáticamente la detección de sonidos en un rango determinado de horas.

Las mayores dificultades en el proceso del desarrollo de este proyecto han sido precisamente los problemas de exceso de consumo de memoria. Este exceso de memoria propiciaba la parada de la captura del audio por la aplicación. Estos problemas han sido solucionados gracias a una optimización de los algoritmos de la aplicación y a la actualización de la versión mínima de *Android* a la 4.0 donde se mejora la gestión de memoria.

En el apartado personal, el alumno ha plasmado y ampliado de una manera práctica algunos conocimientos que ya se poseían, como el uso de la Transformada de Fourier. Y ha servido para incorporar una gran cantidad de conocimientos que no se habían tratado en el grado, como la programación en *Android*, el procesado digital de audio y el uso de patrones de identificación.

7.2 Líneas Futuras

En este proyecto se ha desarrollado una aplicación que incluye por primera vez ayudas a personas con problemas de audición usando su *smartphone*. Aunque la tienda de aplicaciones de *Android* posea un extenso catálogo, hasta ahora no existe ninguna aplicación que pueda ayudar a personas con sordera. Por este motivo, la presente aplicación resulta novedosa y abre la posibilidad de que aparezcan más aplicaciones similares o se consiga una aplicación mejorada de la tratada en este trabajo.

La aplicación no pretende ser lanzada de manera comercial dado que ha sido diseñada para este TFG, pero incorporando las siguientes mejoras podría ser realmente competitiva en la tienda *Android*:

- Mejorar la detección del "timbre de la puerta" y reducir los falsos positivos detallados en el apartado 6.1.6.
- Incorporar más sonidos que pueda detectar la aplicación como un despertador, un fin de lavadora, etc.
- La principal limitación de esta aplicación para ser lanzada al mercado, es que se ha pensado solo para los sonidos detectados en el hogar donde se han realizado las pruebas. Por ejemplo, los sonidos del "fin de microondas" o de un timbre son diferentes según el modelo de timbre o microondas. Una posible solución podría ser incluir un algoritmo de comparación mucho más complejo que contenga las frecuencias características de cada timbre o microondas existentes en el mercado. Otra posible solución, sería realizando una modificación más profunda de la aplicación, dar la posibilidad de introducir sonidos grabados manualmente, que la aplicación analizara sus características y las incorporara al registro de la aplicación, para así poder ser un posible sonido detectado.
- Incorporar una vinculación con redes sociales que tanto se demanda actualmente, es decir, dar la posibilidad de al detectar un sonido registrarlo en *Facebook* o mandar un mensaje a un determinado contacto.

Bibliografía

[1] Alt1040, «El primer teléfono comercial Motorola DynaTAC fue lanzado en el año 1983,» [En línea]. Available: http://alt1040.com/2014/03/motorola-dynatac-30-aniversario-venta. [Último acceso: 26-07-2014].

[2] Wikipedia, «Teléfono inteligente,» [En línea]. Available: http://es.wikipedia.org/wiki/Tel%C3%A9fono_inteligente. [Último acceso: 21-07-2014].

[3] 20 Minutos, «España lidera en Europa en uso de 'smartphones' con un 66% de tasa de penetración,» [En línea]. Available: http://www.20minutos.es/noticia/1900266/0/espana-lidera/uso-smartphones/66-penetracion/. [Último acceso: 15-07-2014].

[4] Es Espectador Digital, «Cuota de merdado de Android,» [En línea]. Available: http://elespectadordigital.com/que-es-la-cuota-de-mercado-de-android/. [Último acceso: 24-08-2014].

[5] PhonicEar, «PocketVib cordless vibrator,» [En línea]. Available: http://phonicear.com/ALD/Assistive_listening_devices/PocketVib.aspx. [Último acceso: 23-07-2014].

[6] Classic Geeks Gadgets, «Motorola PT300 “Handie-Talkie”,» [En línea]. Available: http://www.classicgeekgadgets.com/motorola-pt300-vintage-handie-talkie-radio/. [Último acceso: 24-08-2014].

[7] Wikipedia, «History of mobile phones,» [En línea]. Available: http://en.wikipedia.org/wiki/History_of_mobile_phones. [Último acceso: 24-08-2014].

[8] Altran Smart Society, «Smart Ubiquity: bases para la ubicuidad,» [En línea]. Available: http://altransmart.wordpress.com/2011/11/04/smart-ubiquity-bases-para-la-ubicuidad/. [Último acceso: 24-08-2014].

[9] Tecno Xplora, «El futuro es la velocidad supersónica del 5G,» [En línea]. Available: http://www.tecnoxplora.com/empresas/futuro-velocidad-supersonica_2014072300141.html. [Último acceso: 24-08-2014].

[10] Tecnologia&Gadgets, «Breve historia de los sistemas operativos móviles,» [En línea]. Available: http://tecnologia.hola.com/breve-historia-de-los-sistemas-operativos-mviles/730/. [Último acceso: 24-08-2014].

[11] Xataka, «Diez momentos clave en la historia de la telefonía móvil,» [En línea]. Available: http://www.xataka.com/moviles/diez-momentos-clave-en-la-historia-de-la-telefonia-movil. [Último acceso: 06-08-2014].

[12] M. Gross, «Pensamiento Imaginactivo,» [En línea]. Available: http://manuelgross.bligoo.com/20130219-el-panorama-de-los-6-principales-sistemas-operativos-moviles-en-2013. [Último acceso: 24-08-2014].

[13] Wikipedia, «Sonido,» [En línea]. Available: http://es.wikipedia.org/wiki/Sonido. [Último acceso: 10-08-2014].

[14] Wikipedia, «Historia del registro del sonido,» [En línea]. Available: http://es.wikipedia.org/wiki/Historia_del_registro_del_sonido. [Último acceso: 24-08-2014].

[15] A. d. l. Cuesta, «Recursos para las clases de música,» [En línea]. Available: http://andreadelacuesta.blogspot.com.es/2011/08/historia-del-rock.html. [Último acceso: 24-08-2014].

[16] Hear-it, «Historia de los audífonos,» [En línea]. Available: http://www.hear-it.org/es/Mas-ligeros-mas-pequenos-y-mejores. [Último acceso: 24-08-2014].

[17] VisionFarma, «Historia del Audífono,» [En línea]. Available: http://www.visionfarma.es/content/191/158/453/1/HISTORIA-DEL-AUDIFONO.htm. [Último acceso: 24-08-2014].

[18] Revista Consumer, «Audífonos, el precio de perder oído,» [En línea]. Available: http://revista.consumer.es/web/es/20090401/salud/74682.php. [Último acceso: 24-08-2014].

[19] Blog de Nuevas tecnologías, «Ayudas técnicas a la sordera,» [En línea]. Available: http://abrahambartual.wordpress.com/ayudas-tecnicas-en-sordera/. [Último acceso: 24-08-2014].

[20] F. Zenker, «Ayudas Técnicas para Personas con Discapacidad Auditiva,» Jornadas Técnicas Arquitectura Accesible.

[21] Wikipedia, «Android,» [En línea]. Available: http://es.wikipedia.org/wiki/Android. [Último acceso: 24-08-2014].

[22] Wikipedia, «Núcleo o Kernel (Informática),» [En línea]. Available: http://es.wikipedia.org/wiki/N%C3%BAcleo_(inform%C3%A1tica). [Último acceso: 24-08-2014].

[23] La Biblia del Programador, «Estructura de Android,» [En línea]. Available: http://labibliadelprogramador.blogspot.com.es/2012/09/estructura-de-android.html. [Último acceso: 24-08-2014].

[24] Wikipedia, «Desarrollo de Programas para Android,» [En línea]. Available: http://es.wikipedia.org/wiki/Desarrollo_de_programas_para_Android. [Último acceso: 24-08-2014].

[25] «Eclipse,» [En línea]. Available: http://www.eclipse.org/org/. [Último acceso: 24-08-2014].

[26] Wikipedia, «Entorno de desarrollo integrado,» [En línea]. Available: http://es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado. [Último acceso: 24-08-2014].

[27] JTransforms, [En línea]. Available: https://sites.google.com/site/piotrwendykier/software/jtransforms. [Último acceso: 23-07-2014].

[28] Wikipedia, «Historia de versiones de Android,» [En línea]. Available: http://es.wikipedia.org/wiki/Anexo:Historial_de_versiones_de_Android. [Último acceso: 24-08-2014].

[29] Developer Android, «Dashboards,» [En línea]. Available: https://developer.android.com/about/dashboards/index.html. [Último acceso: 24-08-2014].

[30] Elsatixtech, «Fundamentos de Telefonía, Transmisión de Voz,» [En línea]. Available: http://elastixtech.com/fundamentos-de-telefonia/transmision-de-la-voz/. [Último acceso: 24-08-2014].

[31] Developer Android, «AudioFormat,» [En línea]. Available: http://developer.android.com/reference/android/media/AudioFormat.html#ENCODING_PCM_8BIT. [Último acceso: 24-08-2014].

[32] Wikipedia, «Window function,» [En línea]. Available: http://en.wikipedia.org/wiki/Window_function. [Último acceso: 24-08-2014].

[33] S. Wilson, «CCRMA Stanford,» [En línea]. Available: https://ccrma.stanford.edu/courses/422/projects/WaveFormat/. [Último acceso: 24-08-2014].

[34] Music G, [En línea]. Available: https://code.google.com/p/musicg/. [Último acceso: 24-08-2014].

[35] Source-code, «Java DSP collection,» [En línea]. Available: http://www.source-code.biz/dsp/java/. [Último acceso: 24-08-2014].

[36] S. C. S. S. G. Saha, Indian Institute of Technology, Khragpur, Kharagpur-721 302, India, [En línea]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.138.623&rep=rep1&type=pdf. [Último acceso: 24-08-2014].

[37] GT's Blog, «Silence Removal and End point detection,» [En línea]. Available: http://ganeshtiwaridotcomdotnp.blogspot.com.es/2011/08/silence-removal-and-end-point-detection_29.html. [Último acceso: 24-08-2014].

[38] Aquila DSP, «Aquila Open Soudrce DSP library for C++,» [En línea]. Available: http://aquila-dsp.org/features/. [Último acceso: 24-08-2014].

[39] «FFTW,» [En línea]. Available: http://www.fftw.org/. [Último acceso: 24-08-2014].

[40] Holo Graph Library, [En línea]. Available: https://bitbucket.org/danielnadeau/holographlibrary/wiki/Home. [Último acceso: 24-08-2014].

[41] Wikipedia, «Valor-F,» [En línea]. Available: http://es.wikipedia.org/wiki/Valor-F. [Último acceso: 29-08-2014].

Anexo A: Planificación y Presupuesto

En el presente anexo se va a llevar a cabo un desglose de las tareas que se han realizado a lo largo de este trabajo fin de grado, lo que servirá para realizar posteriormente un cálculo aproximado sobre su coste.

A.1 Planificación

Este trabajo viene precedido de un periodo de Prácticas en Empresa de una duración aproximada de 150 horas, en el Grupo Universitario de Tecnologías de Identificación (*GUTI*) en el que se ha tenido una primera toma de contacto con el lenguaje *Android*: Después de un periodo de documentación se desarrolló una aplicación simple para conocer las funciones básicas del lenguaje *Android* y su IDE. Por lo tanto, este proyecto parte de dichos conocimientos generales de programación en *Android*. Debido a la complejidad de un trabajo de estas características, se ha optado por dividirlo en distintas fases, las cuales se van a comentar a continuación:

Fase 1: Documentación

I. Estudio de la grabación en plataforma Android (5 horas)
II. Preparación de las herramientas de trabajo (5 horas)
III. Estudio de la Eliminación de Silencios (25 horas)
IV. Estudio de Enventanado y Transformada de Fourier (25 horas)

Fase 2: Desarrollo de la aplicación

I. Actividad principal y entorno gráfico (5 horas)
II. Grabación de Sonidos (20 horas)
III. Implementación de la Eliminación de Silencios(30 horas)
IV. Implementación de Enventanado y Transformada de Fourier (25 horas)
V. Identificación de Sonidos (40 horas)
VI. Pantalla de configuración y estadísticas (5 horas)

Fase 3: Pruebas en un dispositivo

I. Pruebas en Sony Xperia Z1 Compact (10 horas)

II. Corrección y depuración (20 horas)

Fase 4: Elaboración de la memoria

I. Redacción de la memoria (80 horas)

II. Correcciones (15 horas)

FASES	HORAS EMPLEADAS
Documentación	60
Desarrollo de la aplicación	125
Pruebas en un dispositivo	30
Elaboración de la memoria	95
TOTAL	**310**

Tabla 6. Desglose de tareas

Trabajando una media de 15 horas semanales este ha sido el diagrama de Gantt del proyecto:

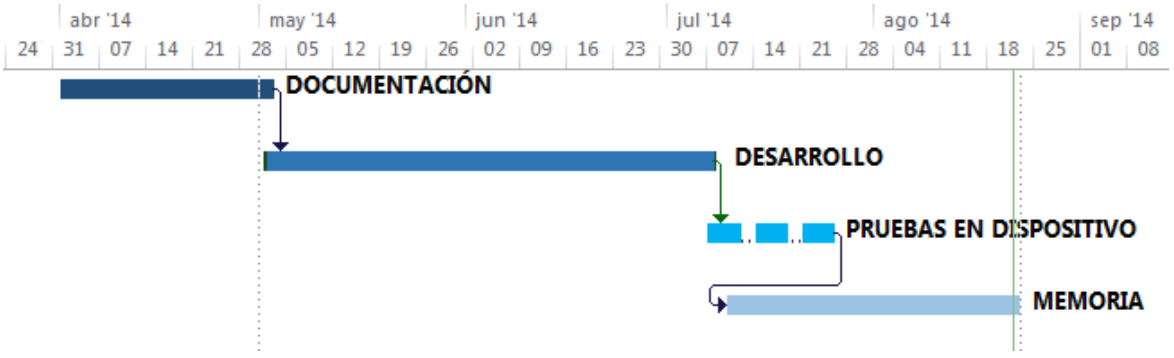


Figura 37. Diagrama de Gantt del Proyecto

A.2 Presupuesto del Trabajo Fin de Grado

A.2.1 Costes materiales

En los costes materiales se incluyen los costes de hardware, los de software y los bienes fungibles. El hardware requerido para el proyecto ha sido un ordenador portátil *Dell* de gama media y un smartphone *Sony Xperia Z1 Compact*. Asimismo, ha sido necesario el uso de herramientas Software como el paquete Office 2010, el IDE Eclipse con el plugin ADT y la librería *JTransforms*, estas dos últimas de licencia libre.

Considerando un periodo de amortización de cada uno de los dos dispositivos y el paquete Office de 3 años, y teniendo en cuenta el tiempo de realización del proyecto es de 5 meses (repartidos en 310 horas), los costes materiales quedan desglosados como se expone en la Tabla 2. El coste de los bienes fungibles también queda detallado en dicha tabla.

CONCEPTO	COSTE UNITARIO (€/Ud.)	COSTE PROYECTO (€)
Ordenador portátil DELL Latitude i5 4gb RAM	700	97.2
Sony Xperia Z1 Compact	450	62.5
TOTAL HARDWARE		**159.7**
Microsoft Office 2010	200	27.8
IDE Eclipse + Plugin ADT	Licencia Libre	0
Librería JTransforms	Licencia Libre	0
TOTAL SOFTWARE		**27.8**
Imprenta	20	20
Internet	30 €/mes	150
TOTAL FUNGIBLES		**170**
TOTAL COSTES MATERIALES		**357.5**

Tabla 7. Costes Materiales del Proyecto

A.2.2 Costes de personal

Para la realización de este trabajo, ha sido necesario el desempeño de un documentalista, un ingeniero y un jefe de proyecto.

OCUPACIÓN	HORAS	PRECIO/HORA	COSTE (€)
Documentalista	30	30	900
Ingeniero	260	50	13 000
Jefe de Proyecto	20	80	1600
TOTAL COSTES DE PERSONAL			**15 500**

Tabla 8 .Costes de Personal del Proyecto

A.2.3 Costes totales

Sumando los dos apartados anteriores, añadiéndole los costes indirectos y el IVA, el coste total del proyecto queda desglosado en la Tabla 4.

CONCEPTO	COSTE (€)
Costes de materiales	357.5
Costes de personal	15 500
Costes indirectos (20%)	3 171.5
Subtotal	**19 029**
IVA (21%)	*3 996.09*
TOTAL PROYECTO	**23 025.09**

Tabla 9. Costes totales desglosados del Proyecto

El coste total del proyecto es de VEINTITRÉS MIL VEINTICINCO EUROS CON NUEVE CÉNTIMOS.

Leganés, 1 de Septiembre de 2014

El ingeniero

Anexo B: Manual de Uso de la Aplicación

En el presente anexo se procederá a relatar brevemente un manual de la aplicación. Aunque en otros apartados como el 4.2.8 del Desarrollo se han podido ver varias capturas de la aplicación y en el 4.2.2 del Diseño de la Aplicación se vea explicado las tareas que se realizan, el presente anexo tiene como objetivo explicar el uso y la funcionalidad de la aplicación a nivel usuario de una manera gráfica con ayuda de capturas de pantalla.

B.1 Inicio de la Aplicación

Una vez instalada la aplicación, esta aparece en el menú de aplicaciones con el nombre "Detector de Sonidos" y un icono en forma de micrófono, pulsando sobre el icono se accede a la pantalla principal.

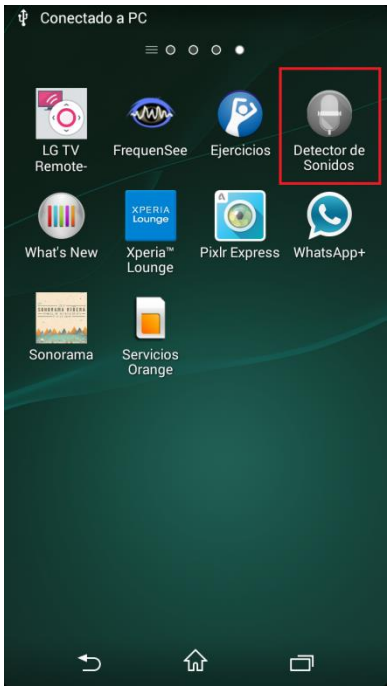


Figura 38. Icono de inicio de la Aplicación Detector de Sonidos

En la pantalla principal, para iniciar la aplicación se pulsará el botón del micrófono, este cambiará su color a rojo para indicar que se están detectando sonidos.

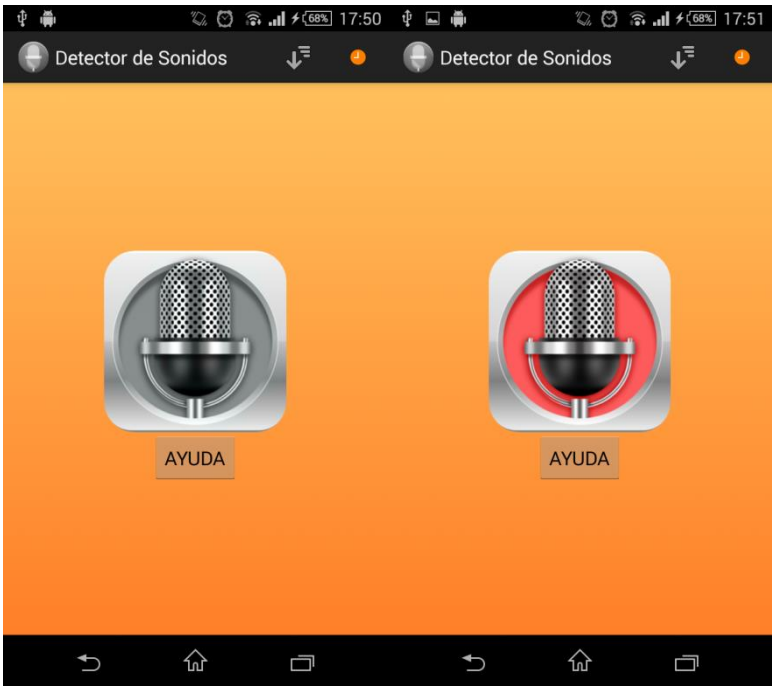


Figura 39. Pantalla principal de la aplicación

A partir de ese momento, se empieza a capturar sonido y se podrá dar a *ATRÁS* y guardar el móvil o realizar otras acciones con él.

Asimismo esta pantalla posee un botón de ayuda donde se relata brevemente el uso de la aplicación detallado en este anexo.

Si la aplicación detecta un sonido el teléfono vibra y aparece un icono representativo del mismo. Los sonidos reconocidos son:

- TELEFONILLO
- TIMBRE DE LA PUERTA (DING- DONG)
- TELÉFONO FIJO
- NEVERA ABIERTA
- FIN DE MICROONDAS

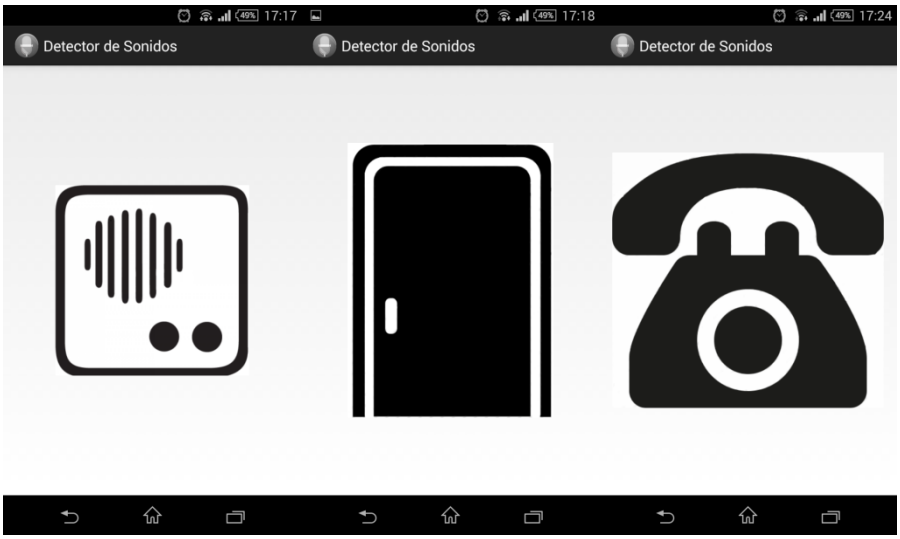


Figura 40. Sonidos detectados: Telefonillo, Puerta y Teléfono Fijo

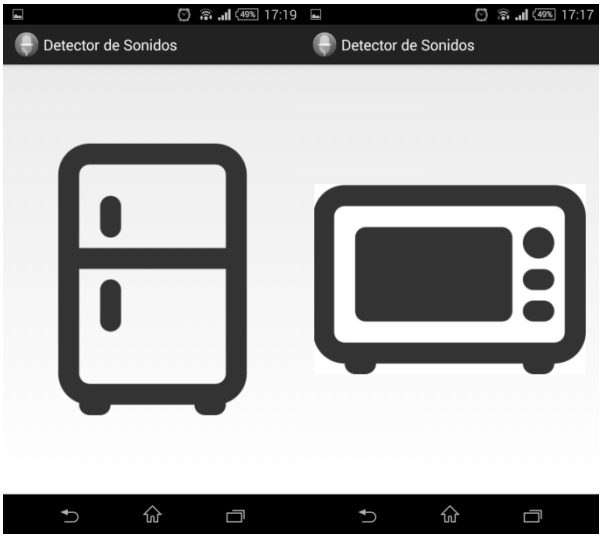


Figura 41. Sonidos detectados: Nevera y Microondas

Cuando se detecta un sonido, la imagen se queda persistente en la pantalla (la pantalla no se apaga) hasta que el usuario pulsa *ATRÁS* y se reanuda la captura de sonidos.

Para terminar con la captura de sonidos, pulsar el micrófono rojo de la pantalla principal.

B.2 Programación de Inicio y Fin Automático

Para programar automáticamente las horas de inicio y fin de la detección de sonidos, pulsar sobre el reloj naranja que aparece en la pantalla principal.

Figura 42. Entrada a la configuración automática de inicio y fin.

Una vez en ese menú se podrá configurar la hora de inicio y la hora de fin.

Figura 43. Menú de configuración automática de inicio y fin.

El menú de *INICIO AUTOMÁTICO* posee los siguientes elementos:

- Un reloj para elegir la hora de inicio automática
- Una casilla de *INICIO MANUAL* que al marcarse inhabilita el reloj y por consiguiente el inicio automático.

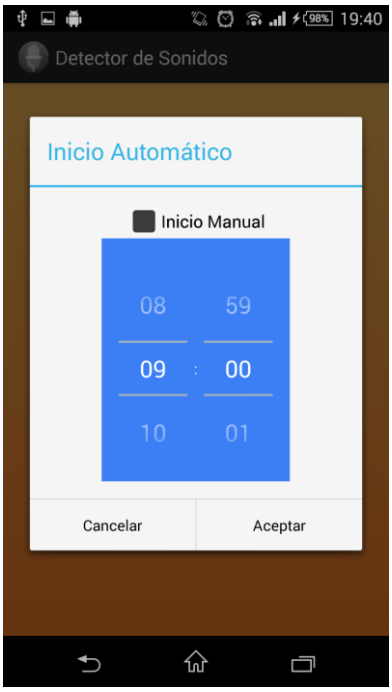


Figura 44. Menú de inicio automático

La configuración del menú de fin automático será idéntica a la de inicio.

El usuario podrá activar el inicio automático, el fin automático o ambos.

Si está activado el inicio automático, pero el FIN no lo está, el usuario parará la identificación manualmente con el botón del micrófono rojo de la pantalla principal.

En cambio, si está activado el FIN automático, pero el INICIO no lo está, será necesario que el usuario inicie la aplicación manualmente para que luego pueda ser parada automáticamente.

B.3 Estadísticas

Para ver las estadísticas que genera la aplicación, pulsar sobre el botón *ESTADISTICAS* en la esquina superior derecha que aparece en la pantalla principal.

Figura 45. Entrada a las Estadísticas de la Aplicación

En este apartado se puede navegar por sus cuatro pantallas deslizando el dedo horizontalmente.

- En la primera pantalla se muestra un logger o registro de la hora y el día en el que el teléfono ha detectado uno de los 5 sonidos.
- En la segunda pantalla aparece el número de veces que ha sido detectado cada uno de los 5 sonidos desde que se ha instalado la aplicación.
- En la tercera pantalla se presenta un gráfico con los datos de la segunda pantalla.
- En la cuarta y última pantalla aparece un *Acerca de…*

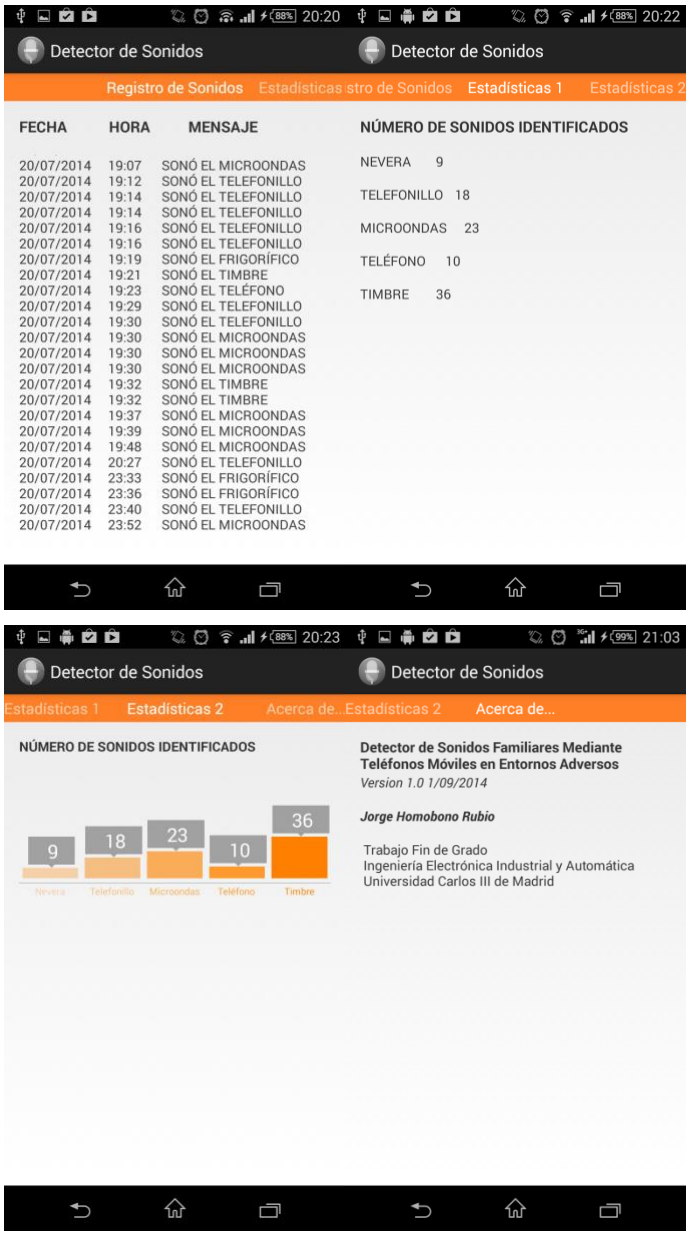


Figura 46. Estadísticas de la Aplicación